Optimal shape design with automatically differentiated CAD parametrisations

by

Orest Mykhaskiv

A thesis submitted to the University of London for the degree of

Doctor of Philosophy

School of Engineering and Material Science

Queen Mary University of London

November 2019

Statement of Originality

I, Orest Mykhaskiv, confirm that the research included within this thesis is my own work or that where it has been carried out in collaboration with, or supported by others, that this is duly acknowledged below and my contribution indicated. Previously published material is also acknowledged below.

I attest that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge break any UK law, infringe any third party's copyright or other Intellectual Property Right, or contain any confidential material.

I accept that the College has the right to use plagiarism detection software to check the electronic version of the thesis.

I confirm that this thesis has not been previously submitted for the award of a degree by this or any other university.

The copyright of this thesis rests with the author and no quotation from it or information derived from it may be published without the prior written consent of the author.

Signature: Orest Mykhaskiv

Date: November 12, 2019

Abstract

Typical engineering workflow for aerodynamic design could be considered as a three-stage process: modelling of a new component in a CAD system, its detailed aerodynamic analysis on the computational grid using flow simulations (CFD) and manufacturing of the CAD component. Numerical shape optimisation is becoming an essential industrial method to improve the aerodynamic performance of shapes immersed in fluids. High-fidelity optimisation requires fine design spaces with many design variables, which can only be tackled with gradient-based optimisation methods. Adjoint CFD can efficiently calculate the necessary flow sensitivities on computational grids and ideally, also CAD parametrisation should be kept inside the loop to maintain a consistent CAD model during the optimisation and streamline the design process. However, (i) typical commercial CAD systems do not offer derivative computation and (ii) standard CAD parametrisations may not define a suitable design space for the optimisation.

This thesis presents an automatically differentiated (AD) version of the open-source CAD kernel OpenCascade Technology (OCCT), which robustly provides shape derivatives with respect to CAD parameters. Developed block-vector AD mode outperforms commonly used finite difference approaches in both efficiency and accuracy. Coupling of OCCT with an adjoint CFD solver provides for the first time a fully differentiated design chain.

Extension of OCCT to perform shape optimisation is demonstrated by using CAD parametrisations based on (a) user-defined parametric CAD models and (b) BRep (NURBS) models. The imposition of geometric constraints, a salient part of the industrial design, is shown for both approaches. Novel parametrisation techniques that can handle components with surface-surface intersections or simultaneously incorporate approaches (a) and (b) for the optimisation of a single component are demonstrated.

The CAD-based methodology is successfully applied for aerodynamic shape opti-

misation of three industrial test cases. Additionally, advantages of the differentiated CAD is showcased for the commonly occurring CAD re-parametrisation and mesh-to-CAD fitting problems.

Acknowledgements

The research presented here was conducted within IODA project[1] - Industrial Optimal Design using Adjoint CFD. IODA is Marie Sklodowska-Curie Innovative Training Network funded by European Commission under Grant Agreement No. 642959. This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT[2].

Firstly, I would like to thank Dr Jens-Dominik Müller for the excellent supervision of my academic work and his coordination of the IODA project. I extend my gratitude to the Queen Mary University of London and to colleagues that I have met here. Besides a productive working environment, our coffee breaks and lunch discussions on aerodynamic design, politics and football will always have a special place in my memory.

It has been a privilege to be involved in the collaborative IODA project, participate in its training, conferences and work alongside other 15 Research Fellows based across Europe. Over the last few years, this enabled joint research with IODA Fellows Mladen Banovic and Salvatore Auriemma resulting in a successful differentiation of the OpenCascade CAD kernel. This would not be possible without the productive secondments to their hosting institutions - the University of Paderborn and Capgemini (OpenCascade).

I was also fortunate to conduct a secondment at Volkswagen as well as a research internship at Autodesk. The invaluable experience gained in the R&D departments in these big organisations had a positive impact on my professional development and I am thankful to people who made it possible. It was great to apply my PhD research in the industrial setting and work on the exciting projects in Automotive Design and Artificial Intelligence.

Finally, I am indebted to my parents, who have been the source of great support

[1] `https://ioda.sems.qmul.ac.uk/`

[2] `http://doi.org/10.5281/zenodo.438045`

and encouragement through the course of this thesis. They always taught me to continue learning and exploring, use every day as a step towards my goals and dreams and most importantly remain optimistic. Interestingly, the same principles can be found in the mathematical formulation of gradient-based optimisation and optimal design, which this thesis is about.

Contents

Nomenclature

Optimisation and AD

$\mathcal{A}$	Optimisation algorithm
α	Design/control variables
f, F	Computer program and its output
$\dot{F}$	Tangent/forward mode derivatives
f_{seed}	Adjoint seed vector
h	Finite difference step size
i	Imaginary number (Complex numbers)
k	Optimisation algorithm iteration
n, m	Number of inputs/outputs
P	Control variables (CAD parameters)
$\overline{P}$	Adjoint mode derivatives
$\mathcal{R}$	Set of real scalars
$\mathcal{R}^n$	Set of n-dimensional real vectors
$S, \overline{S}$	Intermediate primal and adjoint program variables
t	Optimisation algorithm step size
v_{seed}	Tangent seed vector

Flow solver and grid

A	Jacobian matrix $[\frac{\partial R}{\partial U}]$
c_p	Pressure coefficient

D	Mesh deformation matrix
f, g	Tangent/adjoint system vectors $[-\frac{\partial R}{\partial X}, \frac{\partial J}{\partial U}^T]$
J	Cost/objective function
m	Number of mesh points in the grid
m_b	Number of surface mesh points in the grid
$\dot{m}$	Mass flow
μ	Dynamic viscosity
l, v	Tangent and adjoint variables $[\frac{\partial U}{\partial X}, \frac{\partial J}{\partial R}^T]$
L_{def}, a, b, γ	Inverse Distance Weighting parameters
ν	Kinematic viscosity
ν_t	Turbulent kinematic viscosity
$\hat{n}$	Surface normal
Ω	Volume of integration
$\delta\Omega$	Boundary of volume of integration
p_{total}	Total pressure
R	Flow equations residuals
S, dS	Surface of integration
τ	Fluid's stress tensor
$U(\rho, u, p)$	Flow state variable (density, velocity, pressure)
W	Inverse Distance Weighting function
X	Grid mesh points
δX	Perturbed grid points
X_S	Surface grid points
δX_S	Perturbed surface grid points
X_t	Test-point
y^+	Dimensionless wall distance

CAD-based parametrisations

$B(u, v)$	B-spline rational basis function
C	Constraints derivatives matrix
C_s, C_d, C_r	Constraint functions
c_s	TUB Stator blade chord length
$F1, F2$	TUM DrivAer mirror faces
$G0, G1$	Surface continuity
δG	Constraints violations vector
h_s	TUB Stator blade height
$Ker(C)$	Kernel matrix (null space)
n	Number of CAD parameters
m_c	Number of constraints
$N(u)$	B-spline basis function
P	CAD design parameters (NURBS control points)
r	TUB Stator blade LE/TE radius
T_p	Target mesh points (projections)
(u)	1-D parametric coordinate of CAD curve
(u, v)	2-D parametric coordinates of CAD surface
U, Σ, V	SVD matrices
w	NURBS weights
$W1, W2$	TUM DrivAer mirror edge-wires

List of Abbreviations

1-D	One-Dimensional
2-D	Two-Dimensional
3-D	Three-Dimensional
AD	Automatic Differentiation
ADOL-C	Automatic Differentiation by OverLoading in C++
ASO	Aerodynamic Shape Optimisation
BC	Boundary Conditions
BFGS	Broyden–Fletcher–Goldfarb–Shanno (optimiser)
BRep	Boundary Representation
CAD	Computer-aided Design
CAE	Computer-aided Engineering
CAM	Computer-aided Manufacturing
CAx	Computer-aided
CFD	Computational Fluid Dynamics
CP	Control Points
CRM	NASA Common Research Model (aircraft test case)
CSM	Computational Structural Mechanics
ESR	Early Stage Researcher
FD	Finite Difference
GMRES	Generalized Minimal Residual Method
GUI	Graphical User Interface

IDW	Inverse Distance Weighting
ILU	Incomplete LU factorisation
IODA	Industrial Optimal Design using Adjoint CFD (EU Research project)
Im	Imaginary part of a complex number
JT-KIRK	Jacobian-Trained Krylov-Implicit-Runge-Kutta
L-BFGS-B	Limited memory BFGS with box constraints
LE/TE	Leading Edge/Trailing Edge
Ma	Mach number
mgopt	QMUL's in-house solver (early STAMPS version)
MUSCL	Monotonic Upwind Scheme for Conservation Laws
NSPCC	NURBS-based Parametrisation with Continuity Constraints
NURBS	Non-uniform Rational Basis Spline
OCCT	OpenCascade Technology CAD kernel
OOP	Object-Oriented Programming
OSD	Optimal Shape Design
PCA	Principal Component Analysis
PCurve	Parametric Curve on CAD surface
PDE	Partial Differential Equation
QMUL	Queen Mary University of London
RANS	Reynolds Averaged Navier Stokes
RBF	Radial Basis Functions
SIMPLE	Semi-Implicit Method for Pressure Linked Equations
SLSQP	Sequential Least Squares Programming (optimiser)
STAMPS	Source Transformation Adjoint Multi-Purpose Solver
STEP, IGES	CAD file formats
STL	Tessellation/CAD file format
SVD	Singular Value Decomposition

TUB	Technical University Berlin
TUM	Technical University Munich
w.r.t.	with respect to

Chapter 1

Introduction

Ubiquitous digitalisation, among many disciplines, had a profound impact on the engineering design process. In many industries, it resulted in the appearance of so-called CAD/CAE/CAM (CAx) workflows. Computer-aided design (CAD) software is used to create a digital model of a product, specify its shape, material, etc. Benefiting from state-of-the-art modelling and visualisation capabilities provided by most CAD-tools, design teams are able to inspect and manipulate various designs, complicated domains and geometries. Subsequently, the physical performance of these models is exercised within Computer-aided Engineering (CAE) packages, which provide numerical data valuable in further decision-making. The most common use cases include the application of Computational Fluid Dynamics (CFD) (e.g. for aerodynamics, aeroacoustics), Computational Solid Mechanics (CSM) (e.g. for Structural Analysis) and Optimisation. CAE packages leverage on the exponential growth of computing power, advances in mathematical modelling and numerical methods. They use large-scale simulations to predict corresponding complex physics: flow around objects immersed in fluids (CFD), the performance of the objects under certain loads (CSM), etc. Once the engineering objective is achieved, the corresponding CAD model is provided to Computer-aided Manufacturing (CAM) software for production. Here, fast-paced developments in rapid prototyping, 3D printing, advances in machining are revolutionising the way we

make things. Ideally, the corresponding CAx workflow cuts drastically a company's cost and time to deliver a product, where the path from the initial digital drawing to the manufactured component is a few 'clicks' process. However, this requires a very high level of automation and compatibility between all three ingredients of the workflows, maturity and robustness in each of them individually. It is worth noting that the CAx workflow is centred around CAD. Both CAE and CAM assume dependency on the corresponding CAD model and form a 'master-CAD' paradigm.

The field of Optimal Shape Design is a vivid example where this automation is crucial. The ultimate goal of OSD is to change the shape of the object in order to improve its performance. In practice, this means the application of mathematical optimisation algorithms which find optimal values of CAD model parameters. Within a CAD/CAE framework, this requires seamless synchronisation between changes in the CAD model and corresponding CAE calculations.

The idea of CAD in the loop might go even beyond the CAx workflow. Recently, another trend appears to be common in many industries - a set of ideas often referred to as Industry 4.0 or Digital Twin. The twin is a digital representation of the state of the engineering product, while in operation. It could collect and store data on the product performance as well as its multiple individual parts. For instance, the information about the state of blades of a jet engine could be recorded during flights of the aircraft. With advances in data analytics, this opens a wide range of possible applications. The active area of research here is the development of a CAD-like system, capable of adding non-geometric layers of information to the model. Even for this case, the link with typical CAD software seems fundamental.

1.1 Aerodynamic shape optimisation

Aerodynamic shape optimisation (ASO) could be considered as the major branch of OSD. Its goal is to find a shape with the optimal aerodynamic performance. In the context of aeronautics, this could be a shape of an aircraft wing with the

highest lift-to-drag ratio, which also satisfies several geometrical constraints (space inside the wing to store fuel, etc.). Figure 1.1 presents some vivid examples of

Figure 1.1: Top: the updated winglet of Airbus A380plus; bottom-left: F1 front wing; bottom-right: complexity of a turbomachinery engine.

ASO: the modified wing-tip design (winglet) of an Airbus A380plus reduces its fuel burn by up to 4% [1]. The shapes of designed components, their quantity and mutual interaction (multiple vortex generators in an F1 front wing, turbomachinery components with many moving parts) make the problem too complex to be solved based solely on engineering intuition and manual control. But active developments in the fields of CFD, CAD modelling and Nonlinear Optimisation, that are mentioned below, are the key enabling factors to solve these problems with computer simulations. High level of automation between corresponding software components is key to enable efficient design explorations.

1.1.1 Fluid flow modelling

Computational Fluid Dynamics tools were developed rapidly and adopted by many companies over the last few decades. As a rule, CFD solvers implement mathemat-

ical models and numerical schemes that solve partial differential equations (PDEs) on the computational grids. The grids are generated with a meshing software which uses the geometrical description of the input CAD model. CFD software can be considered mature and sophisticated to simulate complex flow physics and accurately predict different aerodynamic cost (objective) functions (e.g. lift, drag, total pressure loss, etc.).

Together with a leap in computing power, models with increased complexity were proposed and tested. Starting with the potential flow and simple 1-D models, now both steady and unsteady Reynolds-averaged Navier-Stokes (RANS) equations are used on a daily basis. Applications of high-fidelity turbulence models such as Large Eddy Simulations and Direct Numerical Simulations, although at much higher computational cost, are also considered. Despite significant efforts to accelerate the solvers on both CPU [88] and GPU architectures [19, 59], the CFD applications remain computationally challenging. Fine resolution of the computational grid is required to capture important flow features. In turn, this has a significant impact on ASO and its turnaround time requirements, as the number of shape configurations that can be tested with the CFD simulations is limited by time-constrained industrial environment.

Adjoint CFD

Sensitivity analysis of CFD models examines ways to compute derivatives of aerodynamic quantities with respect to control variables (e.g. derivatives of the lift force w.r.t. displacement of nodes in the computational grid). Due to high computational cost and large parameter space (thousands of nodes), a straightforward application of finite differences is computationally unaffordable. On the other hand, the adjoint CFD method introduced by Pironneau [103] and developed further by Jameson [63], presents an efficient way of computing CFD model sensitivity information. The approach allows finding all derivatives of the CFD model at a cost similar to the single CFD simulation (primal calculation). The flow sensitivities are

crucial since they provide information on changes in control variables that influence aerodynamics, which can be effectively harnessed in ASO. There are two ways to implement the adjoint method in a CFD code. The continuous approach [41, 65] first derives adjoint equations and then discretises them. Alternatively, the discrete approach can be considered: the flow (primal) equations are discretised and then the differentiation is applied to the discretisation [49]. Both approaches have their advantages and trade-offs [91] connected to the implementation complexity, computational and memory efficiency of the code.

1.1.2 Optimisation: advantages of gradient-based methods

Nonlinear optimisation is a set of methods that can systematically search for a combination of design variables that minimise/maximise given output cost function and also satisfy given constraints. In the context of ASO with CAD, this could be defined as a set of CAD model design parameters that minimise an aerodynamic cost function. One could distinguish two categories of methods often used for shape optimisation. The distinction is based on the order of cost function derivatives used during the optimisation process.

Gradient-free methods

Gradient-free (zero-order) optimisation methods do not require any derivative calculation and use only the current cost function value. Stochastic methods such as Evolutionary algorithms can be used for aerodynamic [122] or structural optimisation [92] but in general require a large number of cost function evaluations. The computational cost of stochastic methods grows dramatically with the size of design space (number of design variables). While reasonable for a handful of design variables, large design spaces might require thousands of function evaluations to find the optimum. Aerodynamic optimisation usually requires very rich design spaces to find optimal flow and effectively tweak small-scale flow features. The convergence to the optimum then requires far too many evaluations of an ex-

pensive computational model such as CFD [80]. This makes use of the stochastic methods computationally prohibitive.

Metamodel-assisted evolutionary algorithms substitute expensive high-fidelity CFD with an approximation - a computationally cheaper surrogate model, which is then used for the optimisation [71, 122]. However, the quality of the optimum here also depends on the number of data-points (CFD calculations) used for the approximation.

The obvious advantage of the stochastic methods is in their global search of the optimum, contrary to the local gradient-based optimisers. Therefore, in the context of ASO, the stochastic methods could be effective in cases when the proposed design was not optimised previously. The global design explorations could be performed by means of low-fidelity models as the initial step before high-fidelity gradient-based optimisation.

Gradient-based methods

This dissertation focuses solely on applications of the gradient-based (first-order) optimisation strategies in ASO. In most general form, the optimisation algorithms minimise a cost function $J(P)$ by updating given design parameters $P = P^0$ using gradient direction. As an example, the steepest descent method iteratively updates control variables with the step-size t:

$$P^{(k+1)} = P^{(k)} - t\frac{dJ}{dP^{(k)}} \,. \tag{1.1.1}$$

The gradient-based algorithms usually require just a few dozens of cost function and gradient evaluations to find the optimum, alleviating computational limitations of the zero-order optimisation methods. Lyu et al. [80] surveyed applications of optimisation methods for aerodynamic design and also show that the gradient-based methods require significantly fewer calculations than the gradient-free methods. Usually, even a limited number of the iterative steps leads to a significant cost function reduction. This is beneficial for ASO turnaround times. Industries tend to optimise already existing components in an incremental fashion rather than

developing completely new designs. In most cases, this also assures proximity of the initial design to the optimal, which fits naturally with requirements of the first-order methods (local optimisation).
The optimisation process requires both cost function value and its derivatives w.r.t. the design parameters. Gradients of the CFD model can be found efficiently with the adjoint method. This dissertation presents ways to compute also CAD model gradients. This enables CFD and CAD software to be used directly in the gradient-based optimisation loop.

1.1.3 Software and open-source: CFD, CAD, Optimisation

Several commercial CAx packages are available and, with certain limitations, could be used for ASO. For the flow calculations, software such as Star-CCM+ [9] and Ansys Fluent [45] is a common choice, they both provide primal and adjoint CFD capabilities. Most commercial CAD packages provide parametric modelling capabilities and were used for gradient-free stochastic aerodynamic optimisation [31]. The CAESES CAD software [2] was developed specifically for the design studies with CFD. It was coupled with Star-CCM+ and used for ASO [124]. Major industrial companies have developed their in-house CFD and CAD solutions [75, 95, 111], designed for their particular needs.
But it's worthwhile mentioning developments in the field of ASO within the open-source community. Access to a source code alleviates limitations of commercial packages, allowing treatment of user-specific problems. Based on collaborative work, the ASO practitioner can develop the optimisation framework without the need for any commercial license.

Automatic Differentiation (AD)

Automatic Differentiation is a collection of mathematical techniques and software tools that can calculate derivatives of a function specified by a computer program [52, 93]. The advantage of AD is that the computed derivatives can be

considered analytic and free of round-off errors. However, to differentiate the program access to the source code is also required. AD is useful in different application domains [14] where gradients are necessary, e.g. sensitivity analysis, gradient-based optimisation. AD remains useful in adjoint CFD code development and recently also has found its way into Machine Learning frameworks [100]. In this work, AD is a key to obtain a gradient-enabled CAD system (Sec. 3.2).

AD tools can be subdivided into two types: source transformation and operator overloading. Source transformation tools parse and analyse the original program and then generate corresponding differentiated code. On the other hand, AD tools with operator overloading make use of object-oriented programming languages features. They redefine all operators present in the code to perform additional derivative calculations. Contrary to source transformation, this approach requires only minor modification of original sources but have its implications on the performance of the code. Several open-source AD tools are available for the source transformation approach (Tapenade [54], OpenAD [116] for Fortran programs) and the operator overloading approach (ADOL-C [123], dco [94], Codipack [107] for C++).

CFD

In recent years several open-source CFD solvers were released. SU2 composes a set of tools for the analysis of PDEs, written in C++ and Python [42, 97]. The software is crafted specifically for adjoint-based ASO and has implementations for both the continuous [41] and the discrete adjoint approach (AD via operator overloading with Codipack [20]). Another popular software with a large user base and a wide range of modelling capabilities is OpenFOAM [66]. The continuous adjoint version of the solver is adopted by Volkswagen [95] and used in collaborative work in this thesis (Chapter 6). Also, the discrete version of adjoint OpenFOAM exists (AD via operator overloading with dco [114, 115]).

The group at the Queen Mary University of London is developing the CFD solver STAMPS (Source Transformation Adjoint Multi-Purpose Solver [88]), which is the

main solver used in the thesis (Subsec. 2.2.1). This is the only open-source discrete adjoint CFD solver that offers the advantages of source transformation AD.

CAD

OpenCascade Technology (OCCT) is a powerful open-source CAD kernel [6]. It provides solutions typical for a CAD system such as surface and solid modelling, data exchange, visualisation. OCCT is used both in an academic and industrial setting, with many applications built around it. FreeCAD is using OCCT as a geometric engine to build an open-source parametric 3-D CAD modeller [3]. The OCCT kernel is enhanced by a convenient graphical user interface (GUI), useful in visualisation purposes.
The open-source nature of OCCT allows direct access to the code and flexibility in development, which in this work enabled successful OCCT differentiation (Sec. 3.2).

Optimisers

There are several well-established optimisation frameworks such as Dakota [17], SciPy [70], pyOpt [101], to name a few. All these tools provide an extensive range of gradient-free and gradient-based optimisation algorithms. In this dissertation, only the gradient-based methods are utilised. Driven by software development assumptions and significant exposure of SciPy ecosystem within the scientific community, the library was chosen as the main optimisation package, complementing several in-house optimisation algorithms (Sec. 2.4).

1.2 Gradient-based optimisation with CAD

1.2.1 CAD-free versus CAD-based methods

In the shape optimisation with CAD and CFD models, computationally efficient gradient-based methods require models' sensitivities to drive the shape changes.

The adjoint method [49, 61] is recognised as the most effective approach, as it allows to compute the CFD sensitivities to an arbitrary number of control variables in a single computation. The CFD sensitivity information could be used to alter the computational grid directly, bypassing its link to CAD. The corresponding CAD-free approaches [108], also often referred to as lattice-based or mesh-based, optimise the positions of mesh points using either a globally interpolated distortion field from radial basis functions [37], an auxiliary grid defining perturbations such as free-form deformation or lattices of Hicks-Henne bumps. In general, the mesh-based approaches use the surface mesh of the CFD grid [64] to impose a displacement. Although the mesh node positions present the richest design space that computational tools can consider, the approach allows surfaces with oscillatory high-frequency noise. This can be addressed by regularisation (smoothing) [65, 68], both implicit [65] or explicit [67] smoothing of gradients or displacements can be applied. However, a major drawback of all the CAD-free methods is that the optimised shape exists only as a mesh. Importing this shape back to CAD is challenging and typically incurs significant approximation. As a consequence, the quality of the optimum is impaired. Another major disadvantage of the CAD-free methods is difficulty in imposing geometric constraints.

As an alternative, CAD-based methods maintain CAD in the design loop and use its parameters as the design variables. Geometric constraints can be defined directly on the CAD geometry and can be incorporated into the optimisation. A projection of the CFD sensitivity into the space of smooth CAD parametrisations filters out aforementioned oscillatory geometries. The main advantage of the CAD-based methods is that CAD model is taken as an input and the optimal CAD geometry is produced as an output. However, to achieve significant aerodynamic improvement, one might be required to define a suitable CAD parametrisation that admits optimal shape deformations (Chapter 4).

The procedure of the gradient-based shape optimisation with CAD in the loop is shown in Fig. 1.2. In the so-called forward pass, the initial geometric parameters

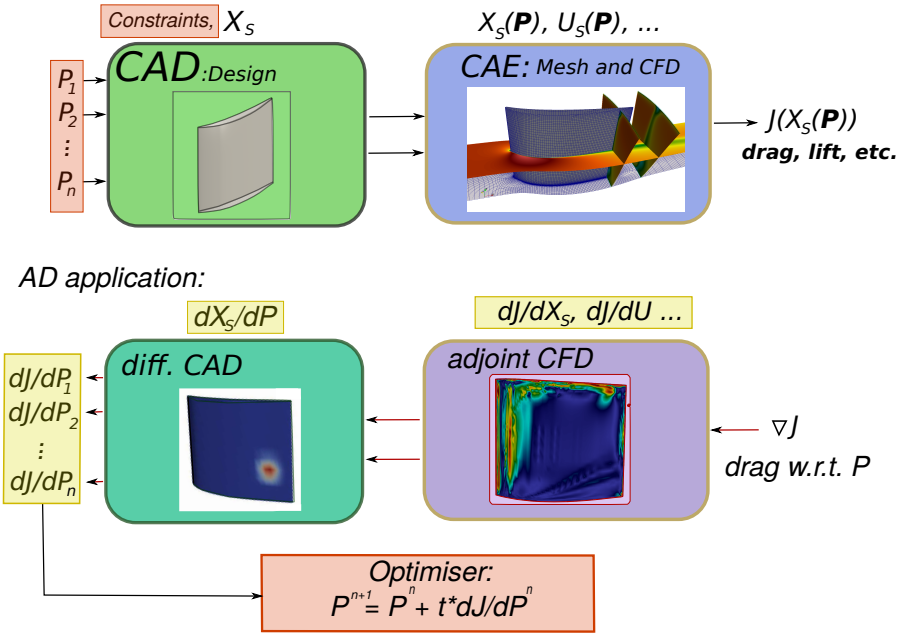

Figure 1.2: Top: Typical CAx workflow for aerodynamics; bottom: gradient-based optimisation of a turbomachinery component with differentiated CAD and CFD in the loop.

P are used to build a CAD model geometry, computational grid and surface mesh points X_S. The CFD solver computes the corresponding flow field U and calculates the aerodynamic cost function J. The propagation of gradients through the outlined system, often referred to as a backward pass, provides the derivatives of aerodynamic cost function w.r.t. the input parameters P. The derivatives can be utilised in the optimisation loop to iteratively update CAD model parameters and the corresponding shape.

1.2.2 CAD gradients (sensitivities): why AD?

In a nutshell, CAD model parametrisation could be considered as a computer program which takes as an input certain parameters (e.g. width, length of a wing) and uses them to execute algorithms that construct geometric surfaces. They describe analytically all surface points of the shape (e.g. pressure and suction surfaces of the

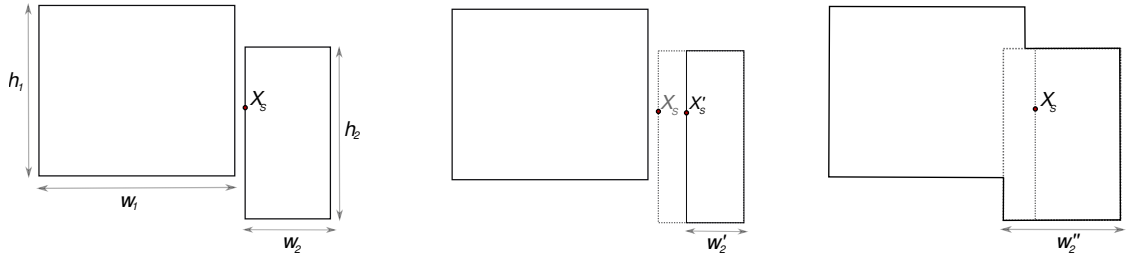

Figure 1.3: Left: initial parametrisation; middle: perturbed CAD model; right: topological changes due to parameter perturbation.

wing) and compose a resulting CAD model. CAD sensitivities (geometric shape gradients) quantify displacements of CAD model surfaces w.r.t. variation of CAD parameters. The shape gradients, necessary for optimisation, are not provided by any of the commercial CAD packages but there are several ways to compute them numerically.

Shortcomings of numerical differentiation

The most straightforward method for computing geometric sensitivities is the application of finite differences (FD). To compute derivative in the surface point X_S w.r.t. all n design parameters P, one would need to generate at least n perturbed CAD models (the process also known as forward differentiation). A first-order approximation of the derivative can be computed with the FD scheme:

$$\frac{dX_S}{dP_j} \approx \frac{X_S(P_j + h) - X_S(P_j)}{h}, \quad j = 1, .., n. \tag{1.2.1}$$

Also, the second-order central-difference FD scheme is often used. Besides $O(n)$ time complexity of the forward differentiation process, the step size h has to be carefully chosen to obtain accurate derivatives, which in turn might require additional calculations.

Figure 1.3 illustrates the process of finding CAD sensitivity in the surface point X_S for a simplified 2-D CAD model. The CAD shape is created with the union of two rectangles which are parametrised with their dimensions (widths w_1, w_2 and heights h_1, h_2). To find the derivative w.r.t. the width of the second rectangle, one

would build the model with perturbed parameter w_2' and measure the distance to the displaced point X_S'. However, certain parameter variations yield topological changes in this model. In Fig. 1.3 (right), the union of rectangles with perturbed parameter w_2'' results in the disappearance of the part of their vertical edges. This situation would require an additional procedure to find the corresponding displacement point X_S''. Although this may seem like a very specific case, similar topological problems such as edge/face renumbering occur quite regularly for more complex CAD models.

Another numerical differentiation technique, the complex-step method, could be employed [84] if the CAD system supports complex numbers arithmetic. The derivatives can be approximated as:

$$\frac{dX_S}{dP_j} \approx \frac{Im(X_S(P_j + ih))}{h} \,. \tag{1.2.2}$$

The approach eliminates truncation errors related to the step-size and, for reasonably small h values, computes derivative approximation to machine precision. It is reported [84] that the complex step method is less efficient than Automatic Differentiation and allows calculation only in the forward mode.

Advantages of Automatic Differentiation

Automatic Differentiation presents an alternative approach for computing CAD derivatives if the CAD source code is available. AD eliminates the drawbacks of the numerical methods mentioned above. It computes derivatives analytically with infinitesimal parameters variations. In the context of the example in Fig. 1.3, the derivative in the point X_S is well-defined. The union operator could be considered as a function which is differentiable almost everywhere, except the value of parameter w_2 where the intersection between two rectangles is about to occur. These cases do not cause problems for AD tools. In the later chapters, we will also see that the intersection algorithms in the OCCT are differentiable. Finally, AD allows computing derivatives with the efficient reverse (adjoint) differentiation

mode, where computational cost is independent on the number of control variables (Sec. 2.1).

CAD systems with derivatives

Robinson et al. [105] apply finite differences to parametric CAD models created in commercial 'black-box' CAD system CATIA V5. To avoid issues with patch renumbering and disappearance due to the finite-size displacements, the geometry is projected onto an STL approximation (tessellation) of the surface. The finite differences of the displacements of grid nodes are evaluated on this STL, which is a computationally expensive process and further affects the accuracy of the gradients. However, the method does allow a user to define design space and constraints through the CAD parametrisation and find necessary shape derivatives (in the paper also referred to as design velocities).

Dannenhoffer and Haimes apply finite differences to the parametric modeller built on top of the OCCT [58]. Where feasible, the authors compute analytical derivatives for simple CAD primitives and shapes (e.g. cube, sphere, etc.). CAD models can be constructed within a convenient GUI (Engineering Sketch Pad), which also has the functionality to visualise the corresponding CAD sensitivities. Recently, the capabilities of the tool were showcased for the aerodynamic shape optimisation [36].

In the work by Xu et al. [126], a small in-house geometric kernel supporting NURBS was automatically differentiated and provides analytical derivatives.

This thesis presents successful differentiation of the entire OCCT CAD kernel with the ADOL-C AD tool [23, 24]. In comparison to the aforementioned approaches, application of AD to OCCT results in a number of advantages: (i) exact and robust analytical shape derivatives can be obtained for complex parametrisations and geometric algorithms; (ii) no finite-size displacement of the geometry is performed during the differentiation; (iii) computational cost of derivatives is much lower than for the FD approaches. This is achieved with ADOL-C features such as forward-

vector and reverse differentiation mode [52].

1.2.3 CAD parametrisations

The definition of the design space is crucial for aerodynamic shape optimisation and CAD-based methods. The optimal result can only be achieved if the existing geometric modes can harness the important aspects of the flow. Therefore, widely used parametric CAD models require from the designer a proper engineering judgement during initial design. To respond to these challenges, several application-specific parametrisation tools were developed [50, 121]. Taking to account extensive engineering experience, these tools allow parametrising the shapes with conventional and intuitive design parameters (trailing/leading edge radius, blade thickness, wingspan, etc.). These parameters can then be varied during the design process. Furthermore, explicit control over the design variables also allows incorporating geometric constraints directly into the parametrisation. These approaches are termed here 'explicit', as they need to be set up manually. However, increasingly 'out of the box' designs are required to work in new configurations or better exploit the interaction between disciplines in multi-disciplinary optimisation. In these situations, a good choice of design parametrisation is often not evident.

Alternatively to the explicit approach, instead of changing the parameters of the model's construction algorithm, one can directly modify the resulting geometric surfaces of the shape [126, 127]. Changes to the BRep data (control point positions and weights of the corresponding NURBS patches) eliminate the initial parametrisation but propose rich design spaces which can be refined adaptively by inserting additional control points. Typically BReps are fine enough to provide a rich set of controls over the surface [68] but the highest frequency modes are adequately low to avoid surface oscillations.

The NSPCC technique [89, 126] (NURBS-based Parametrisation with Complex Constraints) extends this approach from single to multiple patches with fixed edges

and NURBS patch networks with arbitrary topology and order of geometric continuity between the patches. In this thesis, the approach is further extended to support intersecting and trimmed NURBS patches [90]. This NURBS-based method is CAD-vendor independent and requires only a generic CAD-file (STEP, IGES, etc.). The method is referred to as 'implicit' since there is no specific user effort to define the design space.
In this thesis, the differentiated OCCT is extended to support both 'explicit' and 'implicit' parametrisation approaches. The 'explicit' parametrisation is closer to current practice in aeronautics and turbomachinery but may limit the optimum due to restrictive user-defined design space. The alternative 'implicit' approach allows the user to automatically derive a sufficiently rich design space and proposes transitional parametrisation between the parametric CAD-based and mesh-based methods.

1.3 Thesis contributions and structure

This dissertation contributes directly to bridging the gap between CAD, CAE and optimisation tools which can be considered as a stepping stone towards automatic industrial shape optimisation. For the first time, the CAx components are assembled into a fully-differentiated design chain that can be applied to large-scale industrial applications. The developed optimisation framework can compute CFD and CAD derivatives, effectively alter shapes (CAD models) of generic components in order to improve their aerodynamic performance. Novel CAD parametrisation approaches and their applications for the optimal aerodynamic design of several industrial test cases are presented.

Chapter 2 outlines mathematical background of PDE-constrained aerodynamic optimisation. The primal and adjoint CFD solvers and their connection to CAD parametrisations are discussed. The architecture of the developed gradient-based shape optimisation framework allows the coupling of different CFD solvers and

optimisers with a differentiated CAD system.

Chapter 3 shows the successful application of Automatic Differentiation to the OCCT CAD kernel, which subsequently enables the calculation of shape sensitivities. Several ways to compute the derivatives using different AD computing modes are presented and evaluated. A developed block-vector mode facilitates efficient computations for large-dimensional design spaces. To recall typical CAD modelling workflows and OCCT implementation details, prior to this chapter, the interested reader can refer to Appendix A.

Chapter 4 proposes two alternative parametrisation approaches that define different design spaces for CAD-based optimisation. The shape optimisation framework accepts both (i) explicit parametric CAD models built in OCCT and (ii) implicit BRep models available through CAD-vendor neutral files (STEP, IGES, etc.). Both parametrisation techniques offer flexibility in the design space definition and can incorporate optimisation constraints defined directly in the CAD model. Chapter 5 combines the parametrisation approaches (i)-(ii) to optimise different parts of a turbomachinery stator.

Chapter 6 uses the differentiated OCCT to re-parametrise an existing CAD model of an automotive component. The development of an automatic CAD-to-mesh fitting procedure is presented and is shown to be crucial for the construction of efficient design space useful for aerodynamic optimisation of the component.

Finally, a novel parametrisation technique that allows optimising positions of CAD models surface-surface intersections is studied in Chapter 7. The technique is applied to optimise aircraft's wing-fuselage junction.

In summary, the main contributions of the thesis are detailed below:

- Differentiation of complete OCCT CAD software with AD is shown to be feasible, despite the risks and challenges outlined previously in the literature [108]. For the first time, a comprehensive CAD system is equipped with derivatives.

- Development of an efficient block-vector mode of AD for the calculation of geometric derivatives.

- Development and application of shape optimisation framework centred around differentiated OCCT. The advanced architecture of the framework supports seamless integration of different CFD solvers and optimisers.

- CAD parametrisation techniques for industrial optimisation: explicit parametric and implicit NURBS-based models. For both techniques, CAD derivatives are validated, novel methods for incorporation of geometric constraints, their storage and visualisation are presented.

- Use of geometric gradients for automatic CAD model re-parametrisation and fitting. The technique allows the generation of CAD design spaces useful for optimisation and can be applied for commonly occurring reverse engineering (mesh-to-CAD, CAD-to-CAD conversion) problems.

- Aerodynamic optimisation with non-conventional rich design spaces enabled by a) hybrid parametrisations (use of both explicit and implicit parametrisations) and b) parametrisations with intersecting and trimmed surfaces. The techniques present important steps towards automatic shape optimisation with complex CAD models.

1.3.1 Collaborative work

The research presented here was conducted in the collaborative spirit of the IODA project. The differentiation and adaptation of OCCT was performed by three IODA Early Stage Researchers (ESR): Mladen Banovic (University of Paderborn), the present author Orest Mykhaskiv (Queen Mary University of London / QMUL) and Salvatore Auriemma (OpenCascade). M. Banovic was responsible for the injection of ADOL-C into OCCT. S. Auriemma coded parametric model of TUB Stator blade test case in the original OCCT. Their contributions are outlined in

Section 3.2 (See typedef approach) and Subsection 4.2.1, respectively. The use of two CFD codes - STAMPS (in-house QMUL/Pavanakumar Mohanamuraly) and an OpenFOAM-based solver (in-house Volkswagen/Christos Kapellos) - was supported by the ESRs at the corresponding institutions.

Chapter 2

Gradient-based Shape Optimisation Framework

This chapter presents the developed shape optimisation framework and links between its main ingredients: (i) the CAD system which models the shape of the component, (ii) the CFD solver that estimates its aerodynamic performance and (iii) several optimisers that control the shape changes. The chapter highlights the mathematical background of Automatic Differentiation (AD) and its application to the design chain. AD empowered the development of a fully differentiated optimisation framework where the adjoint CFD provides flow sensitivities and CAD system is also equipped with geometric derivatives. In the current gradient-based framework, the software architecture is centred around CAD - different CFD solvers and optimisers can be used.

2.1 Automatic Differentiation

The field of Automatic Differentiation comprises methods and tools that enable derivative computation of computer programs. Contrary to numerical differentiation (e.g. finite difference method, complex step differentiation), AD computes analytical derivatives and does not introduce inaccuracies.

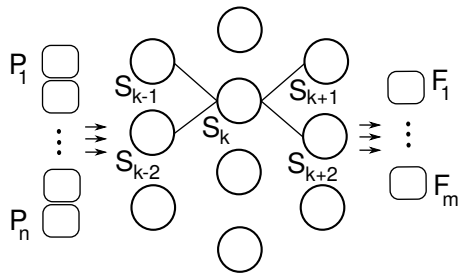

Figure 2.1: Computational graph of a computer program f.

Formally, a computer program $f \in \mathcal{R}^n \to \mathcal{R}^m$ of n inputs P and m outputs F, similarly to a mathematical function, can be represented as follows:

$$F = f(P_1, P_2, ..., P_n)\,. \tag{2.1.1}$$

The program f performs arithmetic and elementary operations on the input, which compute intermediate program variables S_k (k corresponds to the k$^{\text{th}}$ node of the program's computational graph). In this notation, S_k depends on the previous nodes of the graph (e.g. $S_{k-1}, S_{k-2}, ...$), with $(P_1, ..., P_n)$ and $(F_1, ..., F_m)$ as the initial and the final nodes (Fig. 2.1).

Forward (tangent) mode AD tools associate with each S_k its derivative w.r.t. the input variable:

$$\dot{S}_k = \frac{\partial S_k}{\partial P_l}, \quad l \in \{1, ..n\}\,. \tag{2.1.2}$$

The computation starts with defined tangent seed vector (derivatives of the initial nodes):

$$v_{seed} = (0, ..., 1, ..., 0)^T = \dot{P} = (\frac{\partial P_1}{\partial P_l}, ..., \frac{\partial P_l}{\partial P_l}, ..., \frac{\partial P_n}{\partial P_l})^T\,. \tag{2.1.3}$$

Using known derivative expressions for the elementary operations $\frac{\partial S_k}{\partial S_{k-1}}$ [57] and applying the chain rule, AD tools can propagate the derivatives to the output:

$$\dot{S}_k = \frac{\partial S_k}{\partial S_{k-1}}\frac{\partial S_{k-1}}{\partial P_l} + \frac{\partial S_k}{\partial S_{k-2}}\frac{\partial S_{k-2}}{\partial P_l} + ...\,, \tag{2.1.4}$$

$$\dot{F} = (\frac{\partial F_1}{\partial P_l}, ..., \frac{\partial F_m}{\partial P_l})\,. \tag{2.1.5}$$

The latter corresponds to a single column of the Jacobian matrix, obtained by the multiplication of the Jacobian with the corresponding tangent seed vector:

$$\dot{F} = \nabla F \cdot v_{seed} = \nabla F \cdot (0, ..., 1, ..., 0)^T = \nabla F \cdot \dot{P}\,, \tag{2.1.6}$$

$$\nabla F = \begin{pmatrix} \frac{\partial F_1}{\partial P_1} & \cdots & \frac{\partial F_1}{\partial P_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial P_1} & \cdots & \frac{\partial F_m}{\partial P_n} \end{pmatrix}\,. \tag{2.1.7}$$

Setting a different seed vector $v_{seed} \in \mathcal{R}^n$ allows obtaining gradients of the output w.r.t. one of the input parameters. The complete Jacobian can be computed by running the differentiated program (2.1.6) n times with Cartesian basis seed vectors.

The reverse (adjoint) mode of AD first executes the program, records the intermediate variables S_k and stores the computational graph - a procedure often referred to as a trace generation. Afterwards, the computational graph is traversed in the reverse direction. The reverse mode complements intermediate variable S_k with an adjoint:

$$\overline{S}_k = \frac{\partial F_l}{\partial S_k}, \quad l \in \{1, ..m\}\,. \tag{2.1.8}$$

The adjoint variables quantify the changes of the output w.r.t. changes in the intermediate variables [27]. The program derivatives are computed using known adjoint derivatives expressions for the elementary operations $\frac{\partial S_{k+1}}{\partial S_k}$ [57] and by the propagation of the adjoint variables from the output to the input:

$$\overline{S}_k = \frac{\partial F_l}{\partial S_{k+1}}\frac{\partial S_{k+1}}{\partial S_k} + \frac{\partial F_l}{\partial S_{k+2}}\frac{\partial S_{k+2}}{\partial S_k} + ...\,, \tag{2.1.9}$$

$$\overline{P} = (\frac{\partial F_l}{\partial P_1}, ..., \frac{\partial F_l}{\partial P_n})\,. \tag{2.1.10}$$

This allows the computation of one row of the Jacobian matrix:

$$\overline{P} = f_{seed} \cdot \nabla F = (0, ..., 1, ..., 0) \cdot \nabla F = (\frac{\partial F_l}{\partial F_1}, ..., \frac{\partial F_l}{\partial F_l}, ..., \frac{\partial F_l}{\partial F_m}) \cdot \nabla F\,, \tag{2.1.11}$$

$$\overline{P} = \overline{F} \cdot \nabla F\,. \tag{2.1.12}$$

Here, we can seed $f_{seed} \in \mathcal{R}^m$ with a Cartesian basis vector to compute all derivatives for a single output. The full Jacobian can be obtained by running the reverse program (2.1.12) once for each of the m outputs. The reverse mode can be also considered as an implementation of the transposed chain rule (2.1.9), which computes the transposed Jacobian:

$$\overline{P}^T = \nabla F^T \cdot \overline{F}^T \ . \tag{2.1.13}$$

The forward mode is therefore efficient when the number of output variables is higher than the input size ($m > n$). On the other hand, the reverse mode reduces computational complexity dramatically for a program with many input variables and only several outputs ($n > m$). The latter is a common case in aerodynamic applications, where many input grid points or design parameters influence just a few functions of interest. The power of the reverse mode comes at a price of higher implementation complexity (tracing of the computational graph and the bookkeeping procedures are required).

One can also set arbitrary seed vectors v_{seed}, f_{seed} and use AD tools to obtain the corresponding Jacobian-vector products (directional derivatives) efficiently.

For further details on AD theory such as forward and reverse mode, higher order derivatives computations, non-smooth function differentiation or check-pointing, the reader is referred to the reference [52].

2.1.1 AD implementations

Generally, AD tools fall into two categories: source transformation and operator overloading. The approaches have conceptual differences, which in turn also define programming languages they can be used with.

AD with source transformation

The source transformation tools parse the original program and generate additional code which can be used to compute the derivatives, as described above. The program interpretation occurs at compile time. Source transformation AD is

available in a variety of languages such as Fortran [47, 54], C [29], Matlab [30] and others. The approach is most powerful for procedural-style programs, where the parsing of the original source code is relatively simple. This makes the approach useful in numerical computing where the aforementioned programming languages remain popular. Both forward and reverse mode of AD can be supported with the source transformation approach. The evident advantage is in the availability of the differentiated code - both tangent or adjoint sources can be later analysed or amended according to the needs of the user.

AD with operator overloading

Operator overloading is the cornerstone ingredient of object-oriented languages, which could be neatly exploited with corresponding AD tools. There is a variety of AD libraries written for C++ ([107], ADOL-C package [51]), .NET/C# [26], Python [81] that support the concept.

Essentially, all floating-point (primal) variables of a computer program, that contribute to its output, could be re-defined to use AD-specific data type. This new type augments the primal variable with its derivative and implements overloading for all mathematical operations. This involves computation of the primal variables, as in the original code, and calculations of corresponding analytic expressions for the derivatives. Therefore, the instructions for both equations (2.1.1) and (2.1.6) are executed simultaneously. The latter corresponds to the forward propagation of primal and derivative information.

For the reverse mode, the AD tools have a mechanism to record the program's computational graph and intermediate variables to a specific data structure (trace). Afterwards, the trace information is interpreted in the reverse mode, where AD operators propagate adjoint variables as in Eq. (2.1.12).

The great advantage of operator overloading tools is in its natural fit with modern object-oriented codes. As shown in this dissertation, the approach allows differentiation of large industrial-scale codes such as OCCT [24]. It allows preserving

advantages of object-oriented architectures (abstraction, code re-usability through polymorphism) and, depending on implementation, requires minimal changes in the original source code. The memory footprint of these codes might become a bottleneck for programs with deep computational graphs and many intermediate variables which lead to large traces. Chapter 3 addresses this problem in OCCT with the capabilities of ADOL-C.

AD applications

The derivatives of computer programs are essential for gradient-based optimisation in many applications. The necessity of gradients in Machine Learning [27] accelerated development of several computational graph libraries such as Tensorflow [15, 16], PyTorch [100], which also provide gradients with AD. The libraries are mainstream tools for training Deep Learning or probabilistic models with back-propagation (reverse mode AD). Reverse mode AD enables efficient computation of adjoint CFD sensitivity [87, 48, 34] and is useful in aerodynamic design problems [46]. This dissertation benefits directly from two AD tools - Tapenade in the CFD solver STAMPS and ADOL-C in the OCCT CAD kernel.

2.2 Primal and adjoint flow equations

Mathematically, aerodynamic shape optimisation problem could be formulated as follows:

$$\min_{\alpha} \; J(U, X(\alpha)) \,, \tag{2.2.1}$$

$$\text{s.t.} \quad R(U, X(\alpha)) = 0 \,. \tag{2.2.2}$$

Here, the scalar cost function J describes the aerodynamic performance of a given shape e.g. drag, lift, total pressure losses, etc. Expression (2.2.2) denotes the system of steady-state Reynolds-averaged Navier-Stokes (RANS) equations, where the residuals R are driven to zero. J is a function of the state variables U and the mesh coordinates X. The positions of the grid points depend on the shape of the

object and its design parameters α. The design influences $U = U(X(\alpha))$ through the flow constraint R. In the discrete adjoint approach, the values of U and R are considered at the grid points. The sensitivity of the cost function w.r.t. the design parameters can be obtained by the application of the chain rule to equation (2.2.1):

$$\frac{dJ}{d\alpha} = \Big[\frac{\partial J}{\partial X} + \frac{\partial J}{\partial U}\frac{\partial U}{\partial X}\Big]\frac{\partial X}{\partial \alpha} = \Big[\frac{\partial J}{\partial X} + gl\Big]\frac{\partial X}{\partial \alpha}\,, \tag{2.2.3}$$

with g, l defined as $\frac{\partial J}{\partial U}$ and $\frac{\partial U}{\partial X}$, respectively.
Similarly, we can obtain for the residuals:

$$\frac{dR}{d\alpha} = \Big[\frac{\partial R}{\partial X} + \frac{\partial R}{\partial U}\frac{\partial U}{\partial X}\Big]\frac{\partial X}{\partial \alpha} = 0\,. \tag{2.2.4}$$

The latter can be rewritten as a linear system:

$$Al = f\,, \tag{2.2.5}$$

with the Jacobian matrix $A = \partial R/\partial U$ and the source term $f = -\partial R/\partial X$. Solution $l = \partial U/\partial X$ of the system (2.2.5) can be used to compute the sensitivity of the cost function (2.2.3). This 'forward' calculation requires solving linear system (2.2.5) multiple times for each element in X, which results in the computational cost proportional to the size of the mesh.
Fortunately, the more efficient adjoint approach can be applied. Assuming that A is non-singular, equation (2.2.5) becomes:

$$\frac{\partial U}{\partial X} = -\Big(\frac{\partial R}{\partial U}\Big)^{-1}\frac{\partial R}{\partial X} = A^{-1}f\,, \tag{2.2.6}$$

and equation (2.2.3) could be rewritten as

$$\frac{dJ}{d\alpha} = \Big[\frac{\partial J}{\partial X} + \frac{\partial J}{\partial U}A^{-1}f\Big]\frac{\partial X}{\partial \alpha} = \Big[\frac{\partial J}{\partial X} + \frac{\partial J}{\partial U}\Big(\frac{\partial R}{\partial U}\Big)^{-1}f\Big]\frac{\partial X}{\partial \alpha}\,, \tag{2.2.7}$$

$$\frac{dJ}{d\alpha} = \Big[\frac{\partial J}{\partial X} + v^T f\Big]\frac{\partial X}{\partial \alpha}\,. \tag{2.2.8}$$

The adjoint variable v defined as

$$v^T = \frac{\partial J}{\partial U}\Big(\frac{\partial R}{\partial U}\Big)^{-1}\,, \tag{2.2.9}$$

can be obtained from the adjoint system

$$\frac{\partial R}{\partial U}^T v = \frac{\partial J}{\partial U}^T , \tag{2.2.10}$$

$$A^T v = g \, . \tag{2.2.11}$$

The adjoint solution

$$v^T = \frac{\partial J}{\partial R} = \frac{\partial J}{\partial U}\frac{\partial U}{\partial R} \tag{2.2.12}$$

quantifies changes of the cost function w.r.t. the variations in the residual. The adjoint system (2.2.11), contrary to the primal system (2.2.5), does not depend on X and requires only a single linear solve (in the case of the scalar cost function). The flow sensitivity can be computed from the gradient equation (2.2.8) with the computational cost comparable to the primal flow solution. The adjoint flow relationships can be also derived from Lagrange calculus, with v as Lagrange multipliers [112].

In the derivation above we considered only the first term of the sensitivity expression

$$\frac{dJ}{d\alpha} = \frac{dJ}{dX}\frac{dX}{d\alpha} \, . \tag{2.2.13}$$

Since most of the CFD solvers are decoupled from the parametrisation code and take the grid points X directly as the input, the adjoint CFD codes usually compute only $\frac{dJ}{dX}$ - the so-called volume CFD sensitivity. The next subsections describe the projection of this sensitivity vector onto the surface mesh, its coupling with CAD sensitivity matrix $\frac{dX_S}{d\alpha}$ and the propagation of the derivatives through the complete design chain to the design parameters α.

2.2.1 Adjoint CFD with AD: STAMPS solver

STAMPS CFD solver

The group at Queen Mary University of London has developed in-house CFD solver STAMPS (Source Transformation Adjoint Multi-Purpose Solver [88]), which was used as a main solver in this work. STAMPS is based on the group's previous

solver mgopt [53] but additionally includes new features for turbomachinery applications [86] and enables MPI parallelism. The aim of STAMPS is to provide a platform for fully-coupled flow discretisations on unstructured grids. The code is implemented in Fortran, making use of the full language set that is supported by the AD tool Tapenade.

STAMPS uses a typical edge-based vertex-centred finite volume formulation to solve the primal system (2.2.2). Spatially second order convective terms are obtained using MUSCL reconstruction with an approximate Riemann solver based on Roe [106] or AUSM [77] flux schemes. The viscous source terms are obtained using an edge-corrected Green-Gauss formula [38, 55]. After the spatial terms and residuals are obtained, they are integrated in time to drive the solution to the steady-state. STAMPS uses an iterative implicit time integration method based on the JT-KIRK scheme proposed by Xu et al. [128]. JT-KIRK is a multi-grid method using GMRES as a smoother, preconditioned by an ILU decomposition. The algorithm is designed for stability in the discrete adjoints, but numerical experiments also demonstrate that its convergence rate for the primal is superior to other implicit methods. Turbulence modelling is performed with the Spalart-Allmaras RANS model [22]. Further details on the numerical method and implementation can be found in the references [32, 53, 86].

Adjoint CFD in STAMPS

Application of the Tapenade AD tool to the primal steady-state problem (2.2.2) can produce code for the adjoint system (2.2.11) or even the complete CFD sensitivity (2.2.8), if included in the submission to AD. However, the straightforward application of AD to the iterative scheme in the primal solver will generate an iterative scheme that accumulates the adjoint solution from a zero initial field, yielding computational inefficiency in the differentiated subroutines. Fixed-point formulations for the adjoint are typically employed for the continuous adjoint approach and also for hand-differentiated discrete adjoints [49]. Christianson has

formulated AD-based strategies which allow incorporating fixed-point loops into fully AD-produced code [33], which has now been incorporated into a number of AD tools [109]. In STAMPS, fixed-point time-stepping is achieved by submitting only the spatial discretisation to AD, i.e. the code that computes the residual, and calling this in an adjoint iterative driver derived from the primal [32, 34].
Once the adjoint solution is obtained, the sensitivity of the objective (2.2.8) requires also computation of the residual source term $f = -\partial R/\partial X$. While this term does depend on X (large number of control/input variables) - the reverse mode of AD is employed in STAMPS for efficient computation. Followed by the application of AD to mesh morphing algorithm, we can obtain adjoint CFD sensitivity on the design surfaces.

2.2.2 Mesh morphing

In the context of shape optimisation, the changes in the parameters α modify the surfaces of the object. To reflect this variation in the computational grid one could run a re-meshing procedure. Alternatively, mesh morphing techniques can be employed. Geometric (CAD model) and grid displacement can be synchronised with mesh deformation algorithms

$$X = X(X_S(\alpha)) \,. \tag{2.2.14}$$

The changes in the design parameters α can be propagated into the nodal positions of surface mesh point coordinate X_S and then into the volume grid X. This implies that the topology of the surface and the volume mesh remains unchanged. Typically, the following mesh morphing algorithms are used: spring-analogy or Laplacian operators [25], free-form deformation [39], Radial Basis Functions (RBF) [60], linear elasticity [40] or inverse distance weighting (IDW) [125].
The mapping $X_S \rightarrow X(X_S)$ provides a relationship between the surface and volume mesh movement and its gradient $\partial X/\partial X_S$ contributes to the total sensitivity in

Eq. (2.2.13):

$$\frac{dJ}{d\alpha} = \frac{\partial J}{\partial X}\frac{\partial X}{\partial X_S}\frac{\partial X_S}{\partial \alpha} = \frac{dJ}{dX_S}\frac{dX_S}{d\alpha}\,. \tag{2.2.15}$$

We refer to $\frac{dJ}{dX_S}$ as the surface CFD sensitivity, or flow/mesh sensitivity.

It is important to note, that mesh morphing methods are typically well-suited for gradient-based optimisation, which implies moderate and gradual geometric deformations. In these cases, computationally efficient mesh morphing algorithms maintain the quality of the grid and facilitate robustness of the optimisation process. In the case of larger shape deformations, the re-meshing procedure can be performed to obtain a valid mesh. The derivatives of the grid generation algorithm ($X = X_S(\alpha)$) can be then used for the CFD sensitivity calculation [113].

Inverse distance weighting in STAMPS

STAMPS implements the IDW to translate the surface mesh displacement into the volume. The IDW algorithm aggregates boundary displacements of all surface nodes and propagates them into each interior node and hence does not require mesh connectivity information [125]. The IDW is an explicit surface-to-volume interpolation described mathematically as

$$\delta X_i = \frac{\sum\limits_{j\in\partial\Omega} W(||X_i - X_S^j||)\delta X_S^j}{\sum\limits_{j\in\partial\Omega} W(||X_i - X_S^j||)} = \sum_{j\in\partial\Omega} d_i^j\,\delta X_S^j = D\delta X_S \quad i \in \Omega\,. \tag{2.2.16}$$

Here, Ω and $\partial\Omega$ are the volumetric and surface domains. W is the weighting function based on the inverse-distance:

$$W(||X_i - X_S^j||) = \Big[\Big(\frac{L_{def}}{||X_i - X_S^j||}\Big)^a + \Big(\frac{\gamma L_{def}}{||X_i - X_S^j||}\Big)^b\Big]\,, \tag{2.2.17}$$

where L_{def} is the furthest distance from the centroid of the mesh to any of the mesh points. The parameters a, b and γ can be tuned to have a different weighting effect on the nodes in the vicinity and further from the boundary, and are chosen here as $3, 5$ and 0.1, respectively, as recommended in [117]. Equation (2.2.16) can be exactly differentiated (due to its linear form) to yield an explicit surface-to-volume

sensitivity projection:

$$\left(\frac{\partial X}{\partial X_S}\right)^T = \left[d_i^j\right]_{m\times m_b}^T = D^T \,, \tag{2.2.18}$$

where m and m_b are the sizes of the volume and surface mesh, respectively. The IDW is comparable in its final mesh quality to RBF and linear elasticity methods [117]. One must note that IDW is an explicit method, hence it is possible to obtain the surface sensitivity projections only for the desired surface nodes covering the CAD design surface.

2.3 CAD-based design chain with AD

In CAD-based optimisation, the control variables α correspond to the parameters P of a given CAD model. The CAD-based design cycle can be represented as follows:

$$\underbrace{\underbrace{P}_{\text{Parametrisation}} \rightarrow \underbrace{X_S}_{\text{Surface Mesh}}}_{CAD} \rightarrow \underbrace{\underbrace{X_S}_{\text{Surface Mesh}} \rightarrow \underbrace{X}_{\text{Volume Mesh}} \rightarrow \underbrace{U}_{\text{Flow solve}} \rightarrow \underbrace{J}_{\text{Objective}}}_{CFD} \,. \tag{2.3.1}$$

The CAD parameters are used to construct the surfaces of the model in CAD software. The geometry is then used to generate surface and volume mesh points X_S and X, which are used by the CFD solver for the flow simulation and calculation of the cost function. Since in most cases CFD and CAD software are developed independently from each other and possibly using different programming languages (e.g. STAMPS in Fortran, OCCT in C++), AD can be applied to each entity individually. In this context, the two terms in the sensitivity equation (2.3.2) are the flow (CFD) and geometrical (CAD) sensitivity, respectively. The first gradient computes the changes in cost function induced by the infinitesimal movement of the surface mesh points. The second quantifies the displacements of the surfaces mesh points due to the variations in CAD shape.

$$\frac{dJ}{d\alpha} = \frac{dJ}{dP} = \frac{dJ}{dX_S}\frac{dX_S}{dP} \,, \tag{2.3.2}$$

or with reverse mode AD, following the analogy of the transposed chain rule (2.1.13):

$$\frac{dJ}{dP}^T = \frac{dX_S}{dP}^T \frac{dJ}{dX_S}^T . \tag{2.3.3}$$

In this work, CFD sensitivity is always computed with the reverse mode (adjoint), while CAD sensitivity computation is performed with both forward and reverse mode of AD. In the case of forward CAD sensitivity calculation, CFD and CAD sensitivities are assembled into the total sensitivity on the level of surface mesh points. CFD sensitivity of a scalar cost function J in m points X_S:

$$\frac{dJ}{dX_S} = \underset{1\times 3m}{\left[\frac{dJ}{dX_{S,i}}\right]} . \tag{2.3.4}$$

Similarly, CAD sensitivity w.r.t. one of the n design parameters $P = (P_1, ..., P_n)$ could be written as:

$$\frac{dX_S}{dP_j} = \underset{3m\times 1}{\left[\frac{dX_{S,i}}{dP_j}\right]}, \quad j = 1, \ldots, n . \tag{2.3.5}$$

For a 3-D CAD model ($X_{S,i}$ has three components $i = \{1, 2, 3\}$), the gradients listed above are of size $1 \times 3m$ and $3m \times n$ for the CFD and CAD sensitivity, respectively. Once both CFD and CAD sensitivities are available for each surface mesh point X_S, one can assemble the global sensitivity:

$$\frac{dJ}{dP_j} = \sum_{i=1}^{3m} \frac{dJ}{dX_{S,i}} \frac{dX_{S,i}}{dP_j} . \tag{2.3.6}$$

It is important to note, that the adjoint approach and reverse mode differentiation allows obtaining the sensitivities efficiently for large meshes and rich parametrisations ($m, n >> 0$).

CAD with AD

Chapter 3 describes the process of computing CAD sensitivities and details the application of AD to the OCCT CAD kernel. It is possible to obtain parametrisation derivatives dX_S/dP with both forward and reverse mode of AD.

The computed adjoint CFD sensitivity $dJ/dX_S{}^T$ can be provided as the seed vector for the reverse mode routines in the differentiated CAD tool as in Eq. (2.1.12):

$$\frac{dX_S{}^T}{dP}\underbrace{\frac{dJ}{dX_S}^T}_{seed} = \frac{dJ}{dP}^T \quad . \tag{2.3.7}$$

This results in the reverse mode differentiation of the complete design chain (2.3.1):

$$\frac{dJ}{dP}^T = \frac{\partial X_S{}^T}{\partial P}\frac{\partial X}{\partial X_S}^T\frac{\partial R}{\partial X}^T\frac{\partial J}{\partial R}^T . \tag{2.3.8}$$

In certain cases (high memory requirements of the reverse mode in OCCT, Subsection 3.2.3) it is more practical to compute CAD sensitivities with the forward mode. It is then assembled with the CFD sensitivity as in Eq. (2.3.6).

Optimisation loop

Finally, the total gradient can be used to update the parameters of a CAD model:

$$P^{(k+1)} = \mathcal{A}(P^{(k)}, \frac{dJ}{dP}(P^{(k)})) \, , \tag{2.3.9}$$

where $\mathcal{A}$ represents a gradient-based optimisation algorithm on the k^{th} iteration. Figure 2.2 shows the components of the aerodynamic shape optimisation loop. Initially, the computational grid is generated for the baseline CAD model, the positions of surface mesh points on CAD surfaces are determined using a point projection (inversion) algorithm. The CFD sensitivity is computed by running primal and adjoint CFD on the current grid, the CAD sensitivity is computed at the surface mesh points. After coupling of sensitivities, the total gradient is provided to the gradient-based optimiser and the corresponding CAD parameter updates are performed. The CAD model is re-generated with new parameters providing surface movement data (updated positions of surface grid points) to the mesh morphing algorithm. At the end of the cycle, both the CAD model and the corresponding mesh are available. The cycles are repeated until the convergence criteria is met.

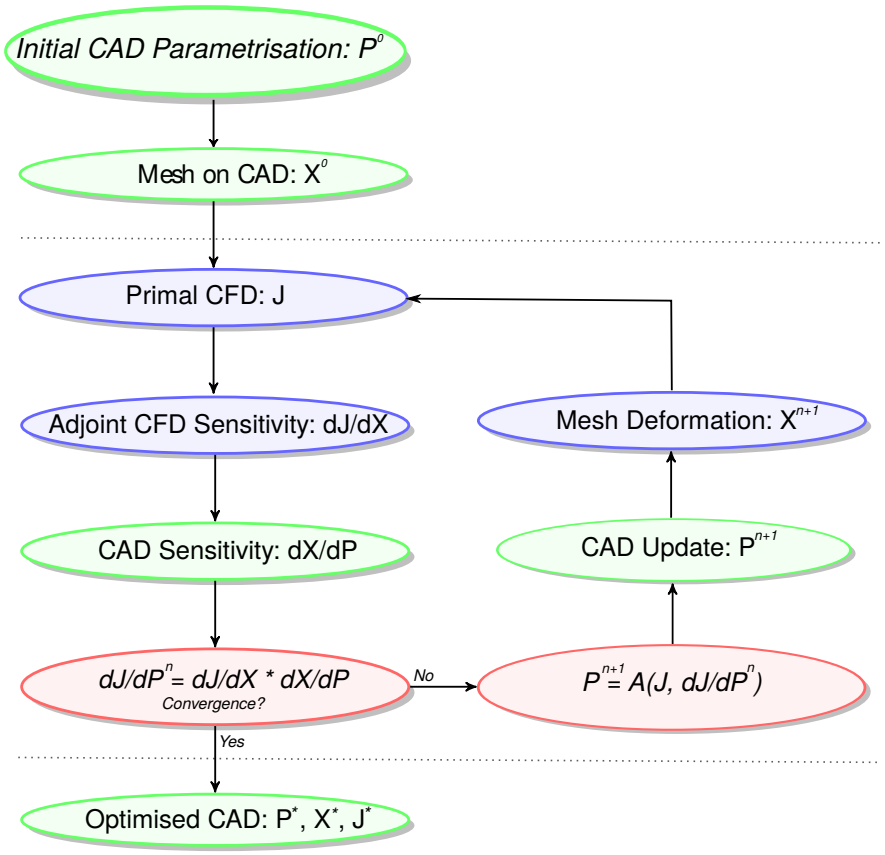

Figure 2.2: Gradient-based optimisation with CAD in the loop: necessary functionality of CAD system (green), CFD solver (blue) and gradient-based optimiser (red).

2.4 Shape optimisation framework

The CAD-based approach for optimisation and availability of the design chain sensitivities allow building the Shape Optimisation Framework centred around the OCCT kernel (Fig. 2.3). In the framework, the differentiated CAD system is augmented by algorithms and data structures for CAD-Mesh manipulations. This helps to maintain correspondence between the CAD model and the computational grid. The framework comprises necessary software components, equipped with gradient calculation capabilities, to perform shape optimisation, as shown in Fig. 2.2. In particular, the object-oriented programming (OOP) features of C++ are used

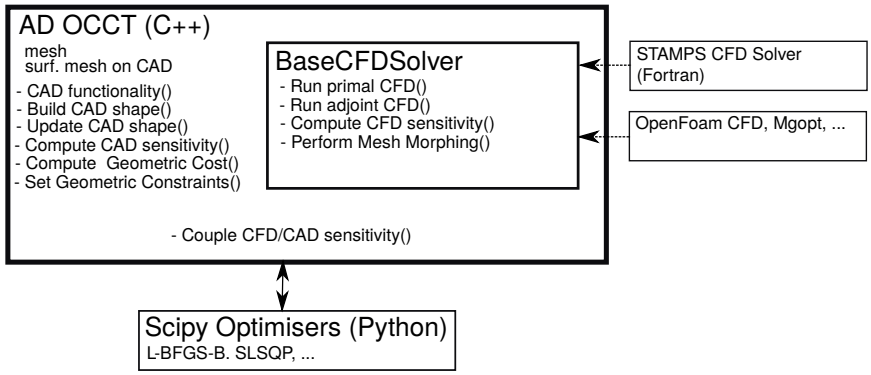

Figure 2.3: Shape Optimisation Framework with coupled CAD, CFD and Optimisation components.

to define an abstract *BaseCFDSolver* class with several purely virtual methods (e.g. definition of methods for computing primal/adjoint CFD flow field, calculation of CFD sensitivity, mesh morphing). To simplify the incorporation of different CFD solvers into the framework, the OOP inheritance mechanism is used - a CFD solver needs to implement the base class methods with the predefined signatures. Therefore, independently of internal CFD solver implementation details, as long as it can compute CFD sensitivity with the adjoint method and perform mesh morphing (Fig. 2.3), it can be easily used. Since the coupling with CAD is implemented on the level of the parent (*BaseCFDSolver*) class, only minimal code changes are required to include a new CFD solver. In this architecture, the base class acts as a low-level API/entry point for CFD tools. To this end, the framework successfully used two in-house CFD solvers developed at QMUL (STAMPS, mgopt) and was coupled with the OpenFOAM-based solver (Chapter 6). The described functionality allows computing gradients of the aerodynamic cost functions w.r.t. the CAD parameters. The framework could also compute purely geometric objective functions (e.g. geometric constraints or distance metrics) and their gradients. This data is then provided to available optimisers.

Besides several in-house optimisation algorithms (gradient descent, projected gradient descent), the framework relies on the optimisation methods in the *SciPy*

library [70]. In particular, the library includes algorithms for constrained optimisation such as *L-BFGS-B* and *SLSQP*. Both algorithms are second order quasi-Newton optimisation algorithms and are well-suited for the problems in shape optimisation. The first method extends the *BFGS* algorithm [78] and allows including bound constraints directly on the CAD parameters ($P_{il} < P_i < P_{ir}$). General constraints (e.g. $c(P) > 0, \quad c \in \mathcal{R}^n \rightarrow \mathcal{R}$) can be imposed with the *SLSQP* algorithm [73, 76]. Binding between Python-based optimiser and differentiated framework (C++) is achieved using *ctypes* - foreign function library in Python.

2.5 Summary

This chapter demonstrated the advantages of the application of Automatic Differentiation to the aerodynamic design chain. AD makes it possible to obtain accurate derivatives of the chain's software components, efficiently providing sensitivities of CFD simulations, mesh deformation and CAD modelling algorithms. Characteristics of the forward and reverse modes of AD and their use for each component of the chain was detailed.

For the first time, the components were assembled into a fully differentiated design chain within the Shape Optimisation Framework centred around OCCT. The framework allows maintaining CAD models during the optimisation process, streamlining industrial CAD-based design workflow. Geometric (manufacturing) constraints can be computed by OCCT and imposed directly on the level of CAD parametrisation. The framework's software architecture was designed to provide flexibility and automation for shape optimisation tasks. Different CFD solvers and optimisers can be used. Applying OOP principles, a new adjoint CFD solver can be quickly added to the framework with minimal code modifications. As the shape optimisation process is often driven by various manufacturing constraints, it is possible to include box constraints (CAD parameter ranges) and more complex non-linear constraints using several available optimisers.

Chapter 3

Differentiated OpenCascade Technology

Geometrical derivatives of a component modelled in a CAD system are necessary for its gradient-based shape optimisation. So far, these derivatives, i.e. shape sensitivities w.r.t. model design parameters, are not provided by any of the existing CAD vendors.

This chapter details the integration of the AD tool ADOL-C into the source code of OCCT kernel. It is the first time that AD was successfully applied to the full CAD system, which consequently enabled shape sensitivities computations. Since OCCT is a complex and large code, several approaches useful for differentiation of OCCT and modern software systems are presented. The developed algorithms can compute shape sensitivities using forward and reverse mode of AD, with the corresponding code snippets included in Appendix B. While the differentiation affects OCCT's data structures and algorithms, the optimal performance of the code is achieved with the developed block-vector AD forward computing mode.

This chapter assumes familiarity of the reader with CAD system workflows and OCCT, which are detailed in Appendix A.

Within IODA project, Mladen Banovic (IODA Research Fellow, University of Paderborn) was the main partner to collaborate on the OCCT differentiation and

contributed with an injection of ADOL-C into the code using the so-called `typedef` approach (Subsec. 3.2.1). Further technical details on OCCT differentiation and its performance can also be found in the joint publication [24].

3.1 Automatic Differentiation with ADOL-C

The choice of AD tool (source transformation, operator overloading) for the differentiation of any code is often driven by the source's architecture and programming language. In the case of object-oriented C++ code like OCCT, the ADOL-C library is a suitable choice. The maturity and flexibility of the library, as well as the collaboration with the ADOL-C development team (University of Paderborn), were essential for the rapid differentiation of the OCCT.

ADOL-C is a software package that facilitates the computation of first and higher order derivatives of vector functions that are defined by computer programs written in C/C++ [52]. The tool was designed to require minimal changes in the original code. It uses the operator overloading concept and hence, contrary to the source transformation approaches, does not generate intermediate source code. AD by operator overloading exploits polymorphism - one of the pillars of the object-oriented programming languages. It allows operators to have different behaviour depending on the argument types they are used with.

ADOL-C enables operator overloading AD with its class/type `adouble`. The class can store variable value (primal value), its derivative and also overloads all differentiable operators. Implementation details for non-differentiable functions such as `abs, min, max, pow, sqrt` and other specific cases can be found in the reference [51].

Listing 3.1 shows a code snippet for the `adouble` class and its multiplication operator. Here, additionally to the primal multiplication, also the corresponding product rule calculation for the derivative is prescribed. In a similar fashion, other mathematical functions and operators are overloaded.

Listing 3.1: Overloading of multiplication operator in ADOL-C (scalar mode).

```
class adouble{
double value;
double ADvalue;
// multiplication overloading
inline adouble operator * (const adouble& a) const {
   myadouble tmp;
   tmp.value = value * a.value;
   tmp.ADvalue = ADvalue*a.value + value*a.ADvalue;
   return tmp;
  }
  ...
  double getADValue(){return ADvalue};
};
```

Hence, the algorithms which use the `adouble` class automatically include directives for both function and derivative calculation. Therefore, the computational graph of the program mimics the chain rule and propagates the derivatives from an input to an output.

ADOL-C provides two kinds of differentiation options:

- *traceless* (**forward (tangent)**: scalar and vector mode)
- *trace-based* (**forward (tangent)** and **reverse (adjoint)**: scalar and vector mode).

Each one implements a different version of the `adouble` class leading to two distinct computational algorithms.
In the traceless option only the forward mode of AD is available. The gradient computation is propagated directly during the function evaluation along with the primal function values. This mode is straightforward to use since every overloaded operator embeds both primal and gradient code in its definition. These derivatives can be accessed by the `getADValue` method of the `adouble` class. Additionally, the traceless `adouble` class can be used in the scalar or the vector mode. The vector mode facilitates simultaneous computation of derivatives w.r.t. several inputs (Listing B.1). With this approach, the algorithm is executed for a vector of variables

rather than several times for each input (scalar mode). The memory requirement for programs that use the vector mode grows linearly with the number of input variables (vector size). However, the corresponding computational time does not follow this trend. ADOL-C vector mode operators exploit compiler vectorisation, which makes the code more efficient than sequential scalar mode execution.

In the trace-based option, operator overloading is used to generate an internal representation (trace) of the function to be differentiated. Then the ADOL-C driver routines such as `gradient, jac_vec, vec_jac, jacobian` are executed on the generated trace to compute the required gradients. In the trace-based option both the forward/tangent and reverse/adjoint mode of AD are available, where reverse mode of AD can dramatically reduce the temporal complexity of the gradient computation.

OCCT was successfully differentiated using both trace-based and traceless options provided by ADOL-C. Hence, it is possible to compute the CAD sensitivities both in the forward and reverse mode of AD.

In this section, we discussed the use of ADOL-C to evaluate first order derivatives necessary in the Shape Optimisation Framework. Further details on ADOL-C capabilities such as higher-order derivatives evaluation, trace-based vector mode, graph colouring methods for sparse Jacobian/Hessian computation, among other topics, can be found in the references [51, 52].

3.2 Differentiated OCCT

3.2.1 Differentiation with type substitution (Typedef approach)

A key ingredient for Automatic Differentiation by operator overloading is the concept of an active variable, which is supported by the `adouble` type in ADOL-C. All variables that may be considered as differentiable quantities at some point during the program execution must be of an active type. Therefore, the integration of

the ADOL-C library into a certain code is done by injection of its specific `adouble` class instead of the native `double` type.

OCCT differentiation is achieved by substitution of all OCCT `Standard_Real (double)` types with `adouble`, using the C++ *typedef* directive - the procedure also later referred to as a *typedef* approach. This kind of integration was challenging due to the size, complexity and deep computational graphs of the object-oriented OCCT code. Although most of `adouble` integration problems were explicitly revealed at compile time, they still required a significant amount of manual code modifications. For instance, several parts of OCCT include a legacy code written in C, some subroutines are used for direct communication with the operating system, the size of original `double` variables is used in assertions, etc. Naturally, such operations are not supported by ADOL-C and in these places the use of `adoubles` was additionally resolved. After successful compilation, also a large number of run-time errors was fixed during the testing phase. The original OCCT distribution includes a test-suite which verifies its functionality. The differentiated OCCT passes successfully 97% of these tests [24], except a few non-geometrical algorithms used for visualisation. This proves non-regression of the differentiated OCCT against the original version (primal calculation) however does not test the values of computed derivatives. Therefore, most algorithms that are used for gradient computation were also compared and inspected against the finite differences result. The mutual agreement between AD and numerical differentiation is demonstrated for the shape optimisation test cases in Chapters 4-7.

The installation process of the differentiated OCCT is similar to the one for the original sources [7]. Additionally, the user is required to provide the location of the ADOL-C libraries. The installation produces a set of compiled libraries that can be used in further application development. The Shape Optimisation Framework uses these libraries for CAD modelling and derivative calculations.

There are two distinct versions of differentiated OCCT that correspond to the trace-based and traceless differentiation option, respectively. Each version uses

corresponding `adouble` implementation. Since the *typedef* approach substitutes all `Standard_Real` variables with `adouble` objects, even those possibly not needed for differentiation, the ideal memory consumption and computational performance is not guaranteed. However, with advances in the AD tool, this inefficiency could be also alleviated. The activity analysis feature, now in development for the trace version of ADOL-C, could build computational graphs (traces) which consider only necessary differentiable quantities.

3.2.2 Overview of alternative differentiation approaches

While the global type substitution (*typedef* approach) affects the computational efficiency of the OCCT kernel, in this thesis, two alternative coding approaches for the differentiation were considered. These approaches maintain the performance of the original OCCT, however, introduce significant modifications to the source code. To test them, a reduced and simplified OCCT kernel was assembled. This compact OCCT version contains only the geometric modules without more complex topological modelling functionality. Although the ADOL-C integration techniques outlined below were discarded for the full OCCT kernel, they can be valuable for smaller codes.

The first approach mimics the behaviour of the source transformation AD tools. The functions subject to differentiation and the subroutines present in the chain rule were duplicated in the sources. In these places, differentiable variables were assigned with the traceless `adouble` type. As a result, the code has a separate original and differentiated function which can be called independently. This approach does not introduce any changes to the primal functionality and allows choosing only necessary differentiable quantities for the type substitution. As a result, the primal CAD modelling performance is unaffected. With this technique, the forward mode AD derivatives of B-spline curves and surfaces w.r.t. their control point positions were successfully obtained, validated and used in geometric gradient-based optimisation. However, this approach faces difficulties when more complex algorithms

are used during CAD model creation. Tracking all subroutines taking part in the construction of the model and their independent manual differentiation proves to be non-practical for codes with deeper computational graphs. The introduced code duplication makes it harder to maintain and update the code.

The second approach examines the use of C++ templates in the differentiation process. The templates allow functions to have different signatures - in our case, `Standard_Real` in the native OCCT sources and `adouble` in the differentiated code. This switches the code from the original to the differentiated version. The coding strategy could be considered as a blend of the first and the *typedef* approach. The templating essentially admits both the differentiated and the original functions, as in the first approach. The necessity of the template declarations in the computational graph can be tracked with the help of the compiler, but contrary to the automatic type substitution (*typedef* approach), this requires intrusive manual replacements of the subroutines' signatures. The forward mode traceless derivatives were also obtained with this approach for the reduced OCCT kernel.

In comparison to the outlined strategies, the chosen *typedef* approach minimises code duplications and modifications. It is the fastest and a global way of integrating ADOL-C into the OCCT code. The great advantage of the approach is that the differentiated code does not differ significantly from the original version. Therefore, developers could use it even without knowledge of AD.

3.2.3 Getting derivatives from OCCT shapes

Appendix A described ways to create CAD models and explore their geometries (shapes) using the original OCCT. As shown in Figure 3.1, this usually implies the propagation of input design parameters through the code. OCCT performs a chain of geometrical operations, defined in the parametrisation, and constructs the final CAD model and its surfaces. In the differentiated OCCT, the `typedef` approach allows using the very same code to achieve this. Additionally, the re-defined `Standard_Real/adouble` variables can be used to propagate also the derivatives.

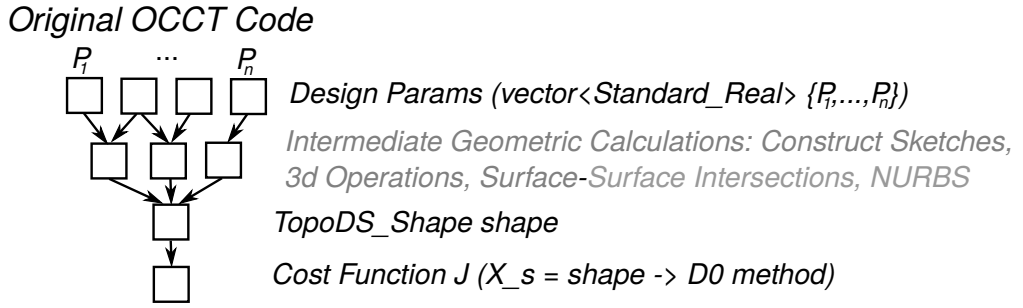

Figure 3.1: Structure of OCCT parametrisation program.

The geometry of a model is characterised by the positions of all its surface points. We defined the CAD sensitivity in the point as the derivative of its position w.r.t. the CAD model design parameter movement. Therefore the surface sensitivities quantify the influence of each design parameter on overall shape and design. The surface point coordinates of a CAD model can be obtained with a standard OCCT $D0$ method. The process of calculating the shape sensitivities and its computational efficiency differs between the traceless and the trace-based version of differentiated OCCT and is outlined below. Here, we assume that the model has n input design parameters and m outputs (e.g. coordinates of m surface points).

Traceless OCCT

1. Scalar mode.

   To calculate the CAD sensitivity w.r.t. one of the design parameters, the corresponding Cartesian basis seed vectors can be set by means of ADOL-C (Listing B.2). The list of all parameters, including the seeded one, can be provided to the OCCT parametrisation code which constructs the final CAD model surface. The computed derivative in the surface point can be accessed with the `getADValue` method of the `adouble` class. Since the traceless ADOL-C implements only the forward mode of AD, we need to perform operations described above n times to compute gradients w.r.t. all design parameters (Fig. 3.2).

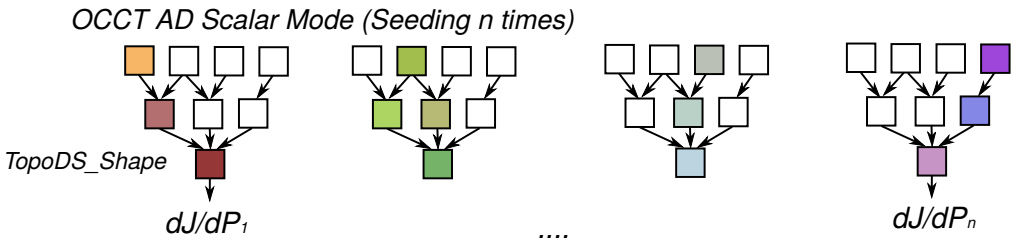

Figure 3.2: Derivative propagation in the forward scalar mode.

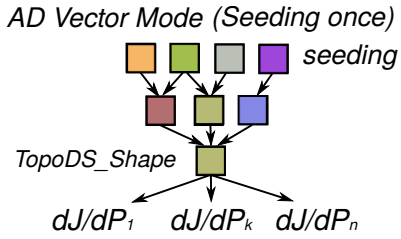

Figure 3.3: Derivative propagation in the forward vector mode.

2. Vector Mode.

   Similarly, the derivatives can be computed with the traceless vector mode. All design variables can be seeded at once using the identity matrix of size $n \times n$ instead of one Cartesian unit vector. Derivative information is propagated for all parameters through the OCCT computational graph simultaneously (Fig. 3.3). The size of the vector (number of directions/parameters) has to be specified in the ADOL-C header or by the dedicated library function. It can not be changed dynamically during the execution of the program. As a result, the size of all `Standard_Real` variables and program's memory consumption grows linearly with a number of directions.

Trace-based OCCT

Trace-based ADOL-C requires a different coding approach for derivative computations. The process can be divided into two stages. Firstly, the code subject to differentiation is declared as the active section with `trace_on/off` ADOL-C commands. Dependencies between the input/output parameters and the values of all `Standard_Real` variables within this section are recorded by ADOL-C to the trace

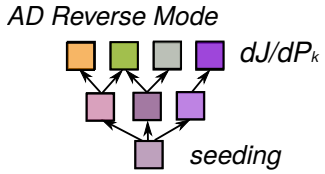

Figure 3.4: Derivative propagation in the reverse mode.

(tape) data structure (Listing B.3). The size of the trace increases with the complexity of the active section. With ADOL-C, one can keep the trace in memory or store larger traces on the hard drive. If necessary, several different traces can be created with distinct tags (identifiers) [51].

In the second stage, ADOL-C routines access the trace with the specified tag and use it to compute derivatives. One can obtain CAD sensitivities with the library routines `jac_vec` and `vec_jac`, which calculate Jacobian-vector product in the forward and reverse mode, respectively (Listing B.4). If the Cartesian basis vectors (tangent) $v_{seed} \in \mathbb{R}^n$ or (adjoint) $f_{seed} \in \mathbb{R}^m$ are provided, the routines compute the columns or rows of the Jacobian matrix. To get all CAD sensitivities (complete Jacobian), the procedure has to be repeated n and m times for the forward or reverse mode, respectively.

For shape optimisation, usually only a handful of cost functions are calculated (e.g. aerodynamic performance, least squares distance; $m = 1$) and hence, the efficient reverse `vec_jac` routine is often used. In the case of aerodynamic optimisation, the CFD sensitivities computed with the adjoint approach can be provided as a seed vector to the CAD sensitivity calculation routine (Figure 3.4, Listing B.4). This ensures the reverse mode differentiation of aerodynamic cost function w.r.t. the CAD parameters including both CFD and CAD software.

Computational Efficiency

The above-mentioned derivative computation methods have differences in implementations and performance. To compare them, the OCCT code analogous to the

`CreateParametricBlade` (Listing A.1) was tested. Eight slices, each constructed on the equidistant plane offsets with 12 control points, were used in the OCCT lofting operation (cross-sectional design approach). The number of slices and control points was chosen to represent a typical effective parametrisation for shape optimisation tasks, where the individual variation of a parameter can cause substantial and, at the same time, local CAD model deformation. As a result, the parametrisation has *n=96* design parameters P that control the final 3-D surfaces. Equally distributed *k=12,000* surface points X_S contributed to the scalar test function ($m = 1$):

$$J = \sum_{j=1}^{k} (X_{S_j}(P))^2 \,. \tag{3.2.1}$$

The performance of the code to evaluate gradients dJ/dP_i was measured for the traceless and trace-based OCCT versions (Fig. 3.5). The traceless version used a vector mode with the number of directions varying from one (scalar mode) to n. The trace-based OCCT in both forward and reverse mode was considered. Additionally, the derivatives were calculated with finite differences using the original OCCT. As the finite differences were computed multiple times for a different number of parameters, the non-smoothness of the corresponding plot is attributed to processes in the operating system, which are not connected with OCCT. The traceless version starts to outperform the finite differences when more than five parameters are used in vector mode calculation. The difference in the performance becomes quite significant when several dozens of parameters are used. As expected, FD performance decreases linearly with the number of directions, contrary to the vectorised ADOL-C implementation (Sec. 3.1). The trace-based forward mode shows similar behaviour to the traceless version, but additional time is spent on the trace's bookkeeping operations. Finally, the trace-based reverse version offers the best computational efficiency, it is independent of the number of input parameters and showcases the strength of reverse (adjoint) Automatic Differentiation. All derivatives are computed in the time comparable to a single derivative computation in the forward mode. Similarly to the trace-based forward version, the reverse

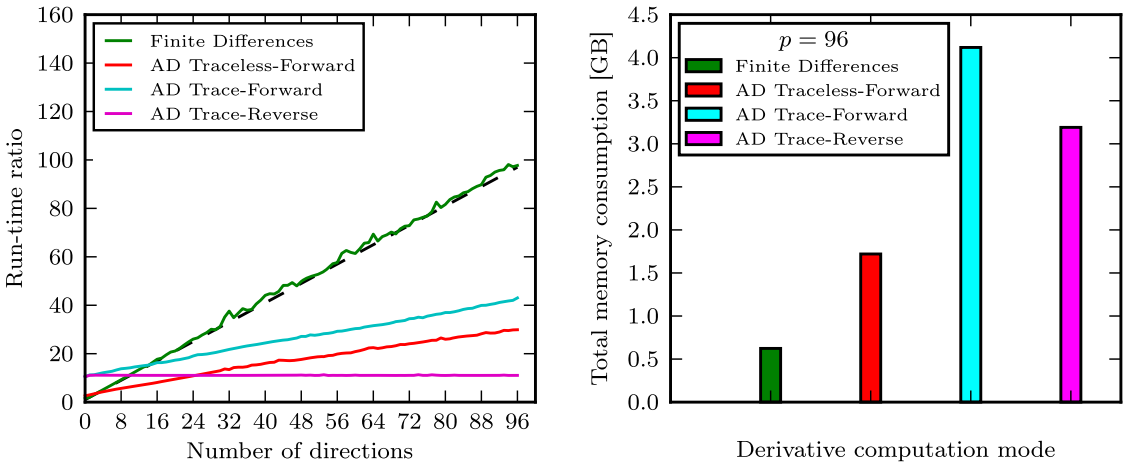

Figure 3.5: CAD derivatives computation as reported by Banovic et al. [24]: run-time efficiency (left) and memory consumption (right).

version is less efficient only if a few parameters are used and the trace-generation time is dominant.

Both traceless and trace-based versions use more memory than the original sources due to the size of the `adouble` class and, in the case of trace-based version, the trace size (Fig. 3.5, right).

3.2.4 Derivatives with efficient block-vector mode

The memory consumption of a differentiated OCCT application is usually proportional to the complexity of the underlying algorithms and the number of used `Standard_Real` variables. In the case of 3-D model construction (extrusion, lofting through slices) with under a few hundred of design parameters - up to 10 Gb of memory could be consumed. Trace-based programs could be even more 'memory-heavy'.

This can, however, become a bottleneck when more complex algorithms are subject to the derivative calculations (e.g. Boolean operations or parametrisations with several thousands of input design parameters as in Chapters 6 and 7). To address this problem, a specific block-vector AD mode based on the traceless OCCT version was developed. The method benefits from the computationally efficient traceless

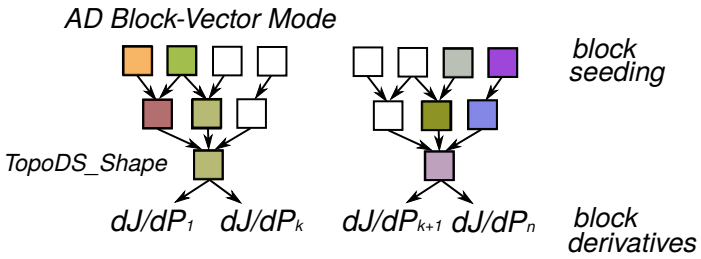

Figure 3.6: Derivative propagation in the forward block-vector mode.

vector mode, while the memory consumption can be adjusted. All design variables are subdivided into blocks of a certain size, which defines the number of directions used by the `adouble` class. Derivatives are computed in the vector mode for the block of variables. Afterwards, another block gets activated and the derivative calculation follows (Fig. 3.6).

The Shape Optimisation Framework contains block-vector mode implementations for derivative calculation, as in Listing B.5. As a result, it is feasible to compute derivatives for very rich parametrisations and complex algorithms. For parametrisations in Chapters 6 and 7, where several thousands of design parameters are used, the size of the block $p = 120$ (number of directions for block-vector mode) has proven to be a good compromise between the memory consumption ($3-5$ Gb) and computational time (few minutes to compute all CAD derivatives).

3.3 Summary

A successful application of Automatic Differentiation to the source code of OCCT was presented in this chapter. For the first time, the full-scale CAD kernel can provide shape sensitivities for various geometric algorithms and CAD modelling workflows. The differentiation of OCCT with ADOL-C presents an essential step towards gradient-based optimisation with the 'master-CAD' paradigm.

Taking into consideration ADOL-C features and the complexity of OCCT sources, three different coding approaches for differentiation were investigated. One finds

that the `typedef` approach is the fastest method of ADOL-C integration into the CAD kernel. This approach substitutes all OCCT numerical variables `Standard_Real` with the ADOL-C `adouble` type, tailored for the derivative computations. The approach requires minimal changes to the evolving OCCT source code, however, also affects its memory usage and computational efficiency. While this is a typical cost for derivative computations with operator overloading, the deficiency can be minimised with the future ADOL-C releases.

The chapter introduced algorithms for CAD model sensitivity calculation, using both forward and reverse modes of AD. For parametrisations with a large number of design variables, developed in this work forward block-vector mode of AD allows users to find a compromise between computational efficiency and memory consumption of OCCT modelling algorithms.

Chapter 4

Parametrisations for Optimisation

A crucial part of shape optimisation is the definition of the design space. Stochastic optimisation methods severely penalise its size and hence require a carefully selected design parametrisation with a few parameters that still allows capturing the optimum. This approach is not suitable for shape optimisation of increasingly complex models. Gradient-based methods, empowered by the differentiated Shape Optimisation Framework, do not suffer from large design spaces requirement and hence allow exploring shapes with extremely rich design spaces.

This chapter proposes two distinct methods for design space definition: (i) explicit parametric and (ii) implicit NURBS-based parametrisations. The first approach relies on aerodynamic and geometric practices to build an effective model in OCCT. Explicit control of design parameters simplifies imposition of manufacturing constraints or even allows embedding them directly into the parametrisation. The second approach does not require coding or pre-processing - it automatically derives the design space from the control points of the commonly used Boundary Representation (BRep) in the form of NURBS patches. The group at QMUL has developed the NSPCC algorithm that supports the formulation of geometric constraints such as continuity across patch junctions. In this dissertation, NSPCC uses derivatives computed by OCCT, includes new manufacturing constraints capabilities such as distance constraints or local curvature. A method to store the

constraint information in the standard CAD files for further inspection or visualisation is presented.

The aforementioned two parametrisation approaches are showcased for the TUB Stator blade test case (Mykhaskiv et al. [89]). Both approaches are generic and, without limitation, can be applied to different geometries. This chapter includes geometric optimisation (parametrisations analysis using CAD model fitting), while the aerodynamic optimisation with CFD is presented in Chapter 5.

4.1 TU Berlin TurboLab Stator test case

The TU Berlin TurboLab Stator (TUB Stator) is a realistic turbomachinery component that could be found in modern jet engine compressors (Fig. 4.1). The purpose of the stator is to turn the incoming flow of 42° whirl angle into an axial direction at the outlet, as shown in Fig. 4.2. The constant mass flow through the stator at 9.5 kg/s is considered. The TUB Stator was proposed as a benchmark test case for aerodynamic shape optimisation with CFD, as a part of optimisation workshop organised by the About Flow EU project [12]. The shape of the stator

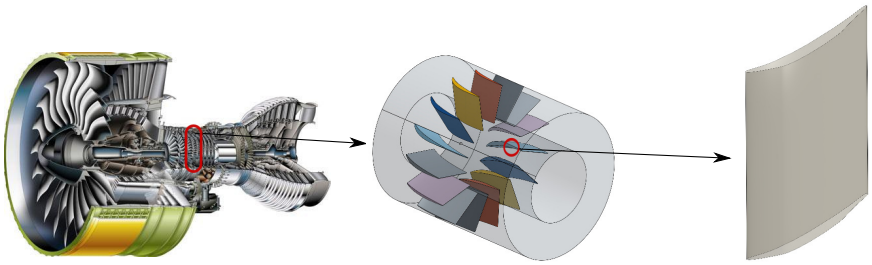

Figure 4.1: Left: Stator blades in jet engine; middle: blades and cylindrical hub/shroud casing; right: individual blade.

blade and its hub (endwall contour) can be modified to improve a geometric (1) and several aerodynamic (2-4) objectives:

1. Minimisation of a least squares distance between the parametrisation and a target blade (Sec. 4.4).

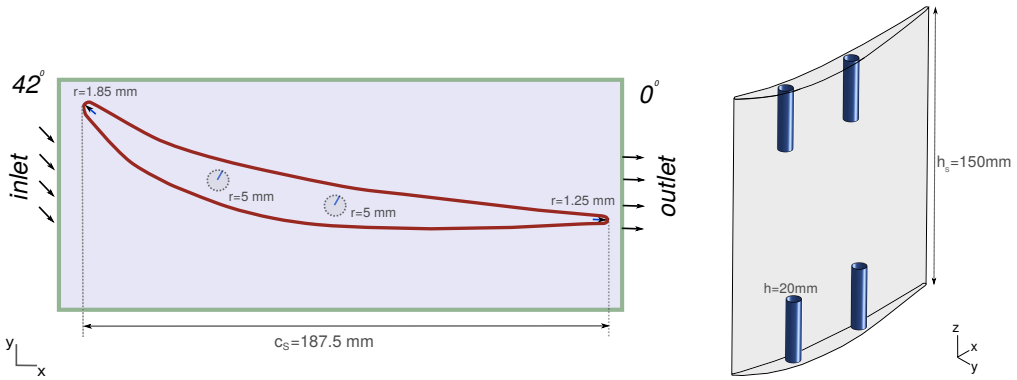

Figure 4.2: Left: TUB Stator blade profile and dimensions; right: BRep model of the TUB Stator blade with cylindrical mounting bolts.

2. Minimisation of total pressure loss between the inlet and the outlet (Ch. 5).

3. Minimisation of flow deviation from the axial direction at the outlet.

4. Minimisation of two previous objectives for three operating points - different inlet whirl angles of 42°, 37° and 47°.

In this dissertation, we consider the first geometric and the second single-objective aerodynamic optimisation problem. In the future, the Shape Optimisation Framework can be also used for the remaining problems 3 and 4.

Geometry and constraints

The baseline CAD geometry of the TUB Stator blade and its cylindrical casing is provided by the workshop (STEP, IGES file format). Changes to the geometry during optimisation are subject to several geometric (manufacturing) constraints:

- Number of blades is fixed to 15: shape optimisation of a single individual blade is considered with periodic boundary conditions.
- Thickness of the blade is restricted by the defined minimum distance between the pressure/suction sides.
- Minimum radius at leading and trailing edge is 1 mm.

- The blade should accommodate four bolts (5 mm radius, 20 mm depth/height; Fig. 4.2) to be mountable to the hub (2 bolts) and the shroud (2 bolts).
- Axial chord of the blade remains constant $c_s = 187.5$ mm.

Further test case details can be found on the workshop homepage [12].

4.2 Explicit parametric design

In this section, the blade is re-parametrised in OCCT using a cross-sectional design approach - a combination of traditional CAD lofting and sweeping (Appendix A). Firstly, a number of cross-sections are generated by the intersection of a swept 2-D blade profile and several horizontal planes. Secondly, the resulting sections are used in 3-D lofting. This allows independent control of each section, creating rich design space. The manufacturing constraints are explicitly defined within the parametrisation and can be provided to any optimiser workflow. The OCCT parametrisation is also fitted to match the original STEP file.
The approach was coded in the original OCCT by Salvatore Auriemma (IODA Research Fellow at OpenCascade) and adapted in the Shape Optimisation Framework by the present author. For the sake of complicity, a brief description of the parametrisation is provided below.

4.2.1 TUB Stator blade parametrisation

The blade parametrisation starts by defining a 2-D profile with B-spline curves, which provide rich parametrisation space [121]. The 2-D blade profile is generated using a camber line B-spline curve characterised by seven control points $(P_1, ..., P_7)$, as shown in Fig. 4.3. Eight reference points $(d1, ..., d8)$ are distributed along the camber line, as shown in Fig. 4.3. The points are also clustered towards the leading and trailing edge (LE and TE) of the camber line. The control points for the suction and pressure side B-splines curves are generated as equidistant offsets of the reference points normal to the camber line (Fig. 4.3). Finally, the suction

and pressure side curves are smoothly joined using the specified radius of curvature satisfying G2 continuity, as explained below.

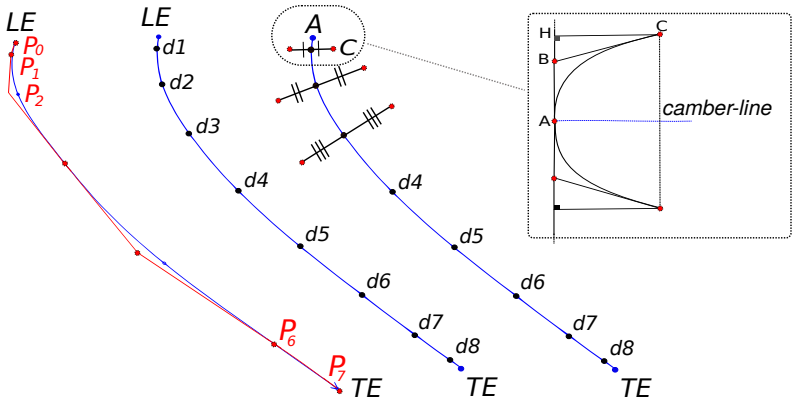

Figure 4.3: Left: Camber line (blue) with corresponding control polygon (red) and reference point distribution; right: construction of pressure/suction control points; imposition of curvature (G2 continuity) at the LE.

For a B-spline curve of degree k (Fig. 4.3) the following holds:

$$AB = \sqrt{curvature(A) \cdot CH \cdot \frac{k-1}{k}}\,, \tag{4.2.1}$$

where AB is the distance between control points A and B and CH is the distance of control point C from the AB line. Therefore, it is possible to impose the curvature in the point A.

This relation is used to distribute three control points of the suction and pressure B-splines with the specified curvatures (radii) at the LE and TE, which are used as design parameters. Thus the imposed leading edge curvature and hence G2 continuity is satisfied.

In summary, the 2-D profile consists of 23 parameters of which, (i) 10 parameters control thickness (2 of them are the radii of TE and LE) and (ii) 13 parameters control the camber line movement (7 control point coordinates in 2-D with fixed axial position of the last control point), as shown in Fig. 4.4.

The 3-D blade parametrisation is based on a lofting, which takes several 2-D slices as input and constructs final B-spline surface using the OCCT approximation algorithm (`BRepOffsetAPI_ThruSections`, Listing A.1). The slices are generated

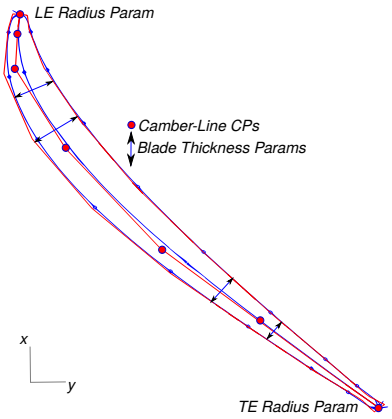

Figure 4.4: Section parameters.

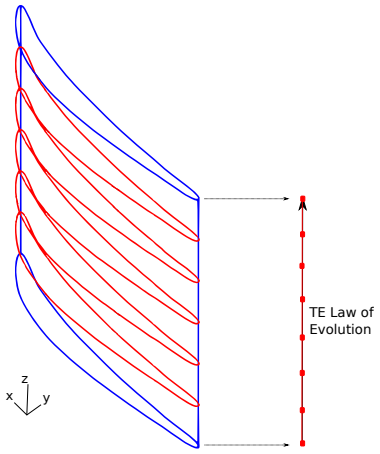

Figure 4.5: Blade skeleton and TE law of evolution in the 3-D domain.

along the blade span: each of 2-D profile parameters is characterised by a law of evolution defined as B-spline curves with 8 control points each. These control points are the design parameters of the optimisation. Their total number is 184 (23×8). An example of the blade construction using several slices is shown in Fig. 4.5.

4.2.2 Evaluation of CAD sensitivities

Shape sensitivities of the explicit TUB Stator blade parametrisation (computed in the differentiated OCCT) were compared with finite differences (FD). As a result, complex geometric algorithms, involved in the cross-sectional blade parametrisation, were tested. A comparison of AD (computed with the forward vector mode) and FD surface sensitivities w.r.t. one of the design parameters (control point of law of evolution) is shown in Fig. 4.6. Mutual agreement is achieved for a decent step size ($h = 10^{-3}$) in the central finite difference scheme $dX_S/dP = (X_S(P+h) - X_S(P-h))/2h$. The figure also shows numerical noise that could be introduced by FD if the step size is chosen poorly ($h = 10e^{-6}$).

Table 4.1 shows the maximum of the absolute value of the difference in the blade's surface sensitivities computed with AD and FD, where the same design parameter

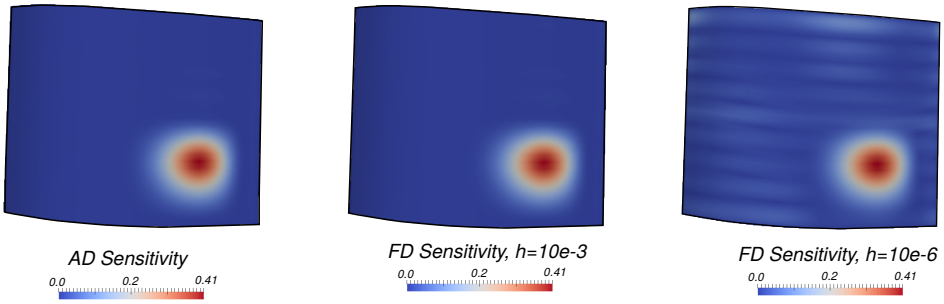

Figure 4.6: Blade sensitivities magnitudes evaluated by AD (left) and FD (middle); FD noise for a small step size h (right).

Step, h	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
AD/FD difference	$8.2 * 10^{-3}$	$9.1 * 10^{-5}$	$1.7 * 10^{-3}$	$9.8 * 10^{-3}$	$1.2 * 10^{-1}$

Table 4.1: Maximal difference between AD and FD sensitivity for different step sizes h.

as in Fig. 4.6 is used. A common trend for FD can be noticed, when round-off errors (high h values) and truncation errors (small h) influence the accuracy of the numerical scheme. The similar trend of the surface sensitivities w.r.t. other design parameters suggests the correctness of AD gradients and therefore ensures robust optimisation.

As defined earlier, CAD sensitivity corresponds to the rate of the surface deformation caused by the infinitesimal perturbation of a CAD design parameter. As expected, OCCT shows high sensitivity values in the areas influenced by the corresponding parameter (e.g. blade thickness or camber line control point position parameters in Fig. 4.7). The CAD sensitivities in the TUB blade parametrisation have a local effect, which can also yield narrow shape deformations useful for optimisation.

In addition to the forward vector mode AD derivatives shown here, Banovic [89] reports also agreement of FD with the OCCT sensitivities computed with the reverse mode of AD.

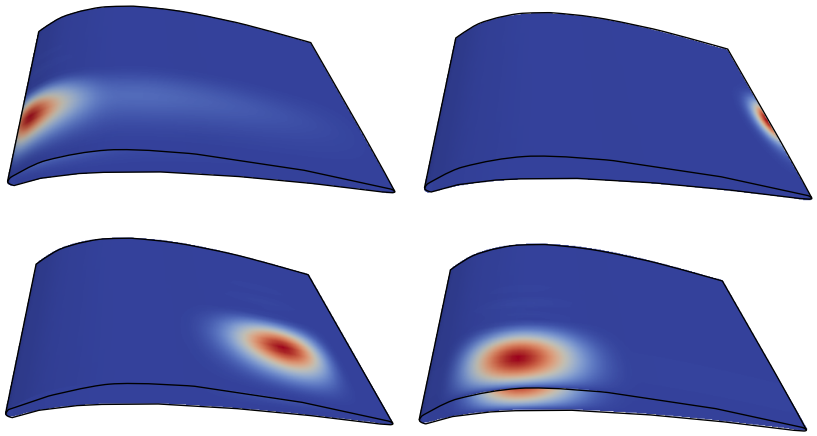

Figure 4.7: Blade sensitivities for several design parameters computed with AD.

4.2.3 Geometric constraints

The parametrisation described above can be used in constrained optimisation and has already several built-in constraints:

- G2 continuity: imposed at the TE, LE and along the sections based on the geometric construction.
- Constant axial chord $c_s = 187.5$ mm: the axial coordinate of the last camber line control point is the offset of the first one: $P_{7_x} = P_{1_x} + c_s$.

The rest of the constraints (minimum blade thickness) could be provided to the constrained optimisation algorithms. For the TUB Stator test case, it is convenient to use an L-BFGS-B optimiser. The range values for the design parameters, which are explicitly available in the parametrisation code, are provided to the L-BFGS-B routines as follows:

- Thickness constraint: the thickness between the suction and pressure surface is approximated using distances between the corresponding pairs of B-spline curves control points, which are also defined as design parameters of the optimisation.

- LE and TE radii: The lower or upper bound values for the radii can be specified as defined by the test case.

The cross-sectional design approach is a powerful method for designing components in OCCT. The parametrisation of each slice with the intuitive design parameters (e.g. blade's camber line and thickness) combined with the laws of evolution admits local and smooth surface deformations useful for shape optimisation. The approach constructs an effective design space, as shown here for the TUB blade, and can be used for other pipe-like geometries [23]. However, for more specific shapes, where generic parametrisation practices do not apply or are not available, a different parametrisation technique might be required. Even if the parametric model is available in a different CAD system, the absence of the parametric standard, as discussed in Subsection A.1.2, could still require a time-consuming reverse engineering task to transfer the model into the OCCT code. To mitigate these problems, another OCCT parametrisation technique was developed - the automatic implicit approach based on BRep.

4.3 Implicit NURBS-based design

The NURBS-based optimisation technique with continuity and geometric constraints (NSPCC approach) was initially proposed in [126] and [127]. In this thesis, the method is extended and automated further. The authors in [126] use a modest in-house CAD kernel, but substituting this with the comprehensive OCCT kernel offers more extensive CAD functionality. Major updates and novelties are related to the refinement of the CAD-space, new constraints capabilities (curvature), recovery of the violated geometrical constraints and the storage of the constraints in standard CAD formats. In the current NSPCC version, OCCT is also used to automate constraint's set-up process. This brings NURBS-based optimisation closer to the industrial workflows and creates an alternative to parametric CAD models optimisation.

4.3.1 NURBS-based design and CAD sensitivities

The advantage of the NSPCC/NURBS-based approach is that in most cases it does not require an understanding of initial parametrisation tree or aerodynamic intuition for the definition of appropriate CAD space. A CAD-vendor neutral Boundary Representation can be retrieved directly from the standard CAD files (STEP, IGES, etc.), which usually contain a collection of NURBS patches. The shape of these surfaces is defined by their control points. Their movement can effectively alter the shape of a modelled component.

NURBS are often used in surface modelling due to the flexibility and locality of its design space. A particular control point and knot distributions allow modelling even 'sharp' features (Fig. 4.8).

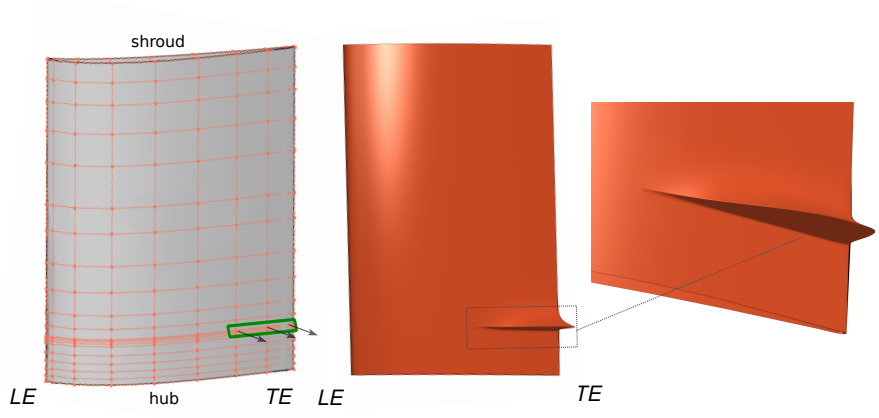

Figure 4.8: TUB Stator blade surface with clustering of Control Points near hub; 'sharp' feature created with the outward movement of several control points in the green box.

Since OCCT is already equipped with an efficient reader of standard CAD formats, its differentiated version allows computing the sensitivity information in any point of the surface with respect to control points perturbations of governing NURBS.

Therefore the CAD sensitivity can be obtained for every surface:

$$\frac{\partial X_S}{\partial P} = {}^{3\,\times\,\mathrm{m}}\Bigg\downarrow \overset{3\,\times\,\mathrm{n}}{\longrightarrow} \begin{pmatrix} \frac{\partial X_{S1}}{\partial P_1} & \frac{\partial X_{S1}}{\partial P_2} & \cdots & \frac{\partial X_{S1}}{\partial P_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial X_{Sm}}{\partial P_1} & \frac{\partial X_{Sm}}{\partial P_2} & \cdots & \frac{\partial X_{Sm}}{\partial P_n} \end{pmatrix}. \tag{4.3.1}$$

Here m and n is the total number of 3-D surface mesh points X_S and control points P, respectively. Moreover, with OCCT one can easily and intuitively refine the design space by adding extra control points with a knot insertion algorithm [102]. This operation does not change the shape or degree of the surface but establishes more control due to the local support properties of the splines. This can be seen in the changing pattern of the CAD sensitivities shown in Fig. 4.9. In the context of aerodynamic optimisation, these very narrow sensitivities could cope better with small flow features not 'visible' for more global parametric sensitivity (Fig. 4.7). At the moment, the refinement is performed manually prior to the optimisation,

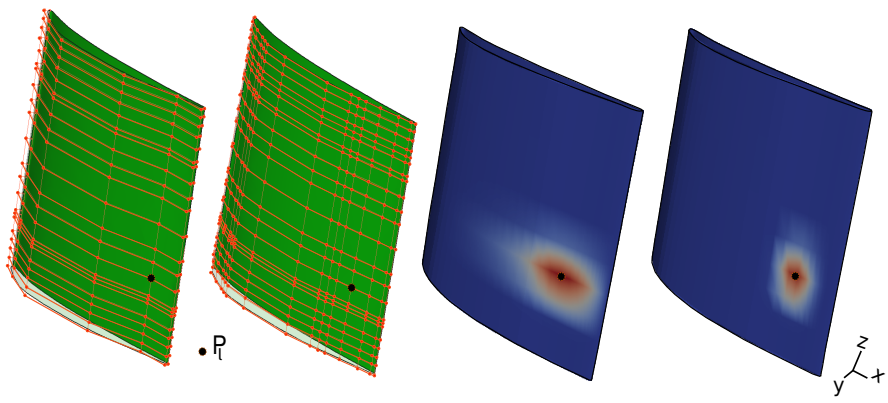

Figure 4.9: Left: initial (9×22) and refined (18×22) control point net of TUB Stator blade; right: CAD sensitivity w.r.t. y-position of a single control point P_{l_y} for initial and refined blade.

but this process can be automated with the CFD sensitivity field as a sensor for refinement [69].

Similarly to the parametric model, the entries of the sensitivity matrix (Eq. 4.3.1) were compared with FD (central difference), providing further assurance in OCCT differentiation. The maximum differences for the blade sensitivities w.r.t. above mentioned P_{l_y} control point position are demonstrated in Table 4.2. Although the

Step, h	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
P_{l_y}, AD/FD diff.	$4.1 * 10^{-11}$	$2.2 * 10^{-10}$	$2.5 * 10^{-9}$	$1.3 * 10^{-8}$	$9.2 * 10^{-7}$
w_l, AD/FD diff.	$3 * 10^{-3}$	$3 * 10^{-5}$	$3.1 * 10^{-7}$	$4.1 * 10^{-9}$	$1.1 * 10^{-8}$

Table 4.2: Maximal difference between AD and FD sensitivity for control point $\|dX_S/dP_{l_y}\|$ and weight $\|dX_S/dw_l\|$ parameters for different step sizes h.

weights of NURBS are not used here for surface modelling, OCCT also provides automatically their sensitivities. Here, the weight w_l of the previously investigated control point l is used for comparison. Due to the difference in the nature of parameters (weight versus control point), the best accuracy of the numerical method is achieved with different step sizes (Tab. 4.2). This shows that for FD, several step sizes might be needed to obtain robust gradients for different parameters. On the contrary, AD gradients are not affected by finite displacements. Similar results can be obtained for other weights and control points.

4.3.2 Geometric constraints

CAD models are usually constructed from multiple adjacent patches. Therefore, modifications of control points individually on patches can violate (i) patch-continuity (holes between the CAD faces, non-smooth shapes) or (ii) other geometrical constraints. The problem can be alleviated by filtering out the shape modes which violate constraints using discrete spaces constructed using test-points [126]. Conceptually, the approach requires that the constraints are satisfied on the particular set of points (test-points) defined on the surfaces of a CAD model.

In the TUB Stator blade test case, a few geometrical constraints are present (LE, TE cross-patch continuity and radii constraints; blade thickness constraints,

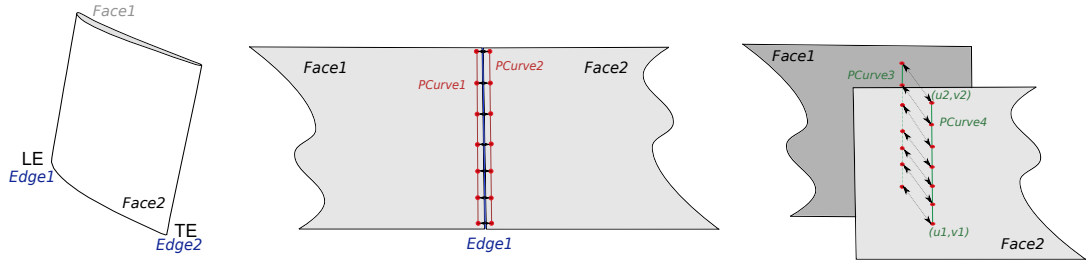

Figure 4.10: Left: TUB Stator blade topology; middle: test-points distribution along *PCurves* of the common edge (G0 continuity, curvature constraint); right: test-points on generic *PCurves* of surfaces (thickness constraint).

Sec. 4.2). Several methods were devised to accelerate and automate the process of test-point distribution. Firstly, we identify topological entities (e.g. edges, parts of surfaces, etc.) necessary for constraint imposition (Fig. 4.10). For instance, to distribute test-points along the leading edge (curvature and G0 continuity constraints), OCCT is used to find two parametric curves (*PCurves*) of the edge on two adjacent faces. Then it is used to uniformly distribute points (in $1-D$ parametric space) along each *PCurve*. As a result, two sets of test-points are generated each belonging to the respective *PCurve*. The test-point pairs along the *edge1* (LE) and *edge2* (TE) are then used to impose continuity and curvature constraints. It is also possible to generate test-point pairs on *PCurves* at arbitrary location on a given patch face. These curves can be defined with the endpoints $(u1, v1)$ and $(u2, v2)$ in the parametric space of the face (Fig. 4.10, right). The generated test-points pairs are then used to impose thickness constraints between the two patches of the Stator blade. The treatment of constraints on a topological level allows storing these *PCurves* in a standard CAD file. This enables visualisation and inspection of the constraints during optimisation. For example, in Fig. 4.11 the pairs of *PCurves* are identically coloured. In addition, the *PCurves* can be stored and visualised as wire-frame objects with vertices as test-points (Fig. 4.11).

Once all necessary test-points are distributed, standard OCCT geometric algorithms (*distance, curvature, normal, etc.*) can be used to compute the following

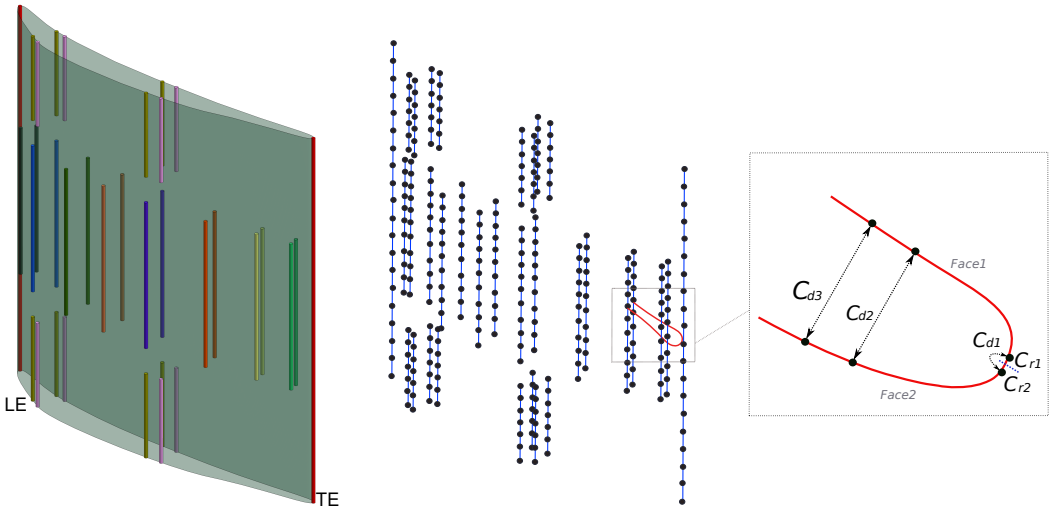

Figure 4.11: Left: Constraints position visualisation from the STEP file; middle-right: constraint functions on the test-points pairs.

constraints:

- Distance constraints

  To fix the distance d_r between two test-points (X_{t1}, X_{t2}), the following function is constructed:

  $$C_d = distance(X_{t1}, X_{t2}) - d_r = 0\,. \tag{4.3.2}$$

  For the TUB blade, this constraint is used to ensure G0 continuity (with $d_r = 0$) and the constant axial chord length. Similarly, the minimum thickness (T_{min}) constraint, which is required in the middle of the blade and for the bolts, corresponds to an inequality constraint and is represented with:

  $$C_d = 1 - min\big(1, \frac{distance(X_{t1}, X_{t2})}{T_{min}}\big) = 0\,. \tag{4.3.3}$$

- Radius of curvature constraint

  OCCT allows users to compute the minimum and maximum curvature at any point of the surface. Therefore, the radius in the test-point corresponding to the TE and LE can be calculated as $r = 1/curvature(X_{t1})$. Constraint

function bounding the minimum radius value to (r_{min}) is:

$$C_r = 1 - min\big(1, \frac{r}{r_{min}}\big) = 0\,. \tag{4.3.4}$$

- Smoothness constraint

  G1 continuity can be imposed as:

$$C_s = normal(X_{t1}) \times normal(X_{t2}) = 0\,. \tag{4.3.5}$$

The *min* operator in equations (4.3.3) and (4.3.4) is used to 'activate' inequality constraint if it gets violated, and 'deactivate' it (constraint value is zero) otherwise. The differentiated OCCT is used to assemble derivatives of all constraint-functions into the constraint matrix:

$$C = \frac{\partial C_{d,s,r}}{\partial P} = {}_{3\times n}\Bigg\downarrow \overrightarrow{\qquad\qquad m_c \qquad\qquad} \begin{pmatrix} \frac{\partial C_{d1}}{\partial P_1} & \frac{\partial C_{d2}}{\partial P_1} & \dots & \frac{\partial C_{r1}}{\partial P_1} & \frac{\partial C_{r2}}{\partial P_1} & \dots & \frac{\partial C_r}{\partial P_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C_{d1}}{\partial P_n} & \frac{\partial C_{d2}}{\partial P_n} & \dots & \frac{\partial C_{r1}}{\partial P_n} & \frac{\partial C_{r2}}{\partial P_n} & \dots & \frac{\partial C_r}{\partial P_n} \end{pmatrix}. \tag{4.3.6}$$

Here, m_c corresponds to the number of all above-mentioned constraints.

NSPCC and projected gradient method

During optimisation, an update δP from a feasible design P^k ($C^k_{d,s,r} = 0$) should not cause violations of the constraints ($C^{k+1}_{d,s,r} = 0$). Linearisation of the constraint functions yields:

$$C^{k+1}_{d,s,r} \approx C^k_{d,s,r} + \sum_{i=1}^{N} \frac{\partial C_{d,s,r}}{\partial P_i}(P_i^{k+1} - P_i^k)\,, \tag{4.3.7}$$

$$C\delta P = 0\,. \tag{4.3.8}$$

Therefore, the allowable design update should lie in the null space of the linearised constraint matrix C. The null space can be computed numerically using the Singular Value Decomposition (SVD) [126]:

$$C = U\Sigma V^T\,, \tag{4.3.9}$$

where U and V are the unitary matrices of size $m_c \times m_c$ and $3n \times 3n$, respectively and Σ is a diagonal matrix with r non-zero singular values (r is also the rank of the matrix C). The last $3n - r$ columns v of the matrix V are known to span the null space of C and are noted here as a kernel matrix $Ker(C)$. The allowable design update can be then obtained as a linear combination of the columns v with coefficients $\delta\alpha$:

$$\delta P = \sum_{k=r+1}^{3n} v_k \delta\alpha_k = Ker(C)\delta\alpha \, . \tag{4.3.10}$$

The last relationship can be used in the optimisation algorithm:

$$P^{(k+1)} = P^{(k)} - t \cdot Ker(C)\Big[(\nabla J)Ker(C)\Big]^T \, , \tag{4.3.11}$$

where

$$\nabla J = \frac{\partial J}{\partial X_S}\frac{\partial X_S}{\partial P} \, . \tag{4.3.12}$$

Equation (4.3.11) could be considered as a projected gradient method with the allowable descent direction and the step size t. This ensures that during the optimisation the control points perturbations are in the null space of the constraint matrix, i.e. the control points are modified without violating the constraints (at least for infinitesimal step size).

Number of test-points (constraints)

Since defined *PCurves* are located on NURBS surfaces, the satisfaction of a constraint in a limited number of test-points along a given *PCurve* can guarantee that the constraint is satisfied everywhere on that *PCurve*. The number of test-points needed to achieve this depends on the properties of the corresponding NURBS surface (degree, number of control points, local support properties, etc.). In this work, the number of the required test-points is determined manually before the optimisation. After a sufficient amount of test-points is distributed, the rank of the constraint matrix C (computed by the SVD) does not increase with additional test-points. The excessive test-points, therefore, do not influence the allowable design update and are filtered out by the Eq. (4.3.10) of the NSPCC algorithm [130].

Constrained non-linear optimisation

In addition to the projected gradient method, the Shape Optimisation Framework allows using different optimisation algorithms. The same test-points approach can be used to define constraints directly on the surfaces of the component. The optimisers, such as SLSQP, which is supported in the Framework, requires the values of constraints and their gradients w.r.t. design parameters. The constraint functions and the constraint matrix (Eq. 4.3.6) can then be provided to the optimisation algorithm. Both equality (e.g. constant chord length) and inequality constraints (e.g. minimum distance between test-points) can be used.

4.3.3 Recovery of constraints

Due to non-linearity of constraints (G1, curvature) and inequality constraints (some constraints are inactive) they could be violated after the finite step in Eq. (4.3.11). To overcome this, the continuity recovery method proposed previously in [126] was extended to all type of constraints.

Firstly, the violated constraints are indicated ($|\delta G_{d,r,s}| = |C_{d,r,s}| > \epsilon$) and are placed into a violation vector $\delta G_{violated} = (\delta G_1, ..., \delta G_{N_{violations}})$. The constraint matrix is decomposed into two matrices $C = C_{violated} \cup C_{satisfied}$ with columns entries corresponding to the violated or satisfied constraints, respectively. Afterwards, the necessary control points update (recovery) should satisfy:

$$C_{violated}\delta P_{upd} + \delta G_{violated} = 0\,, \tag{4.3.13}$$

which also has to satisfy the rest of the constraints:

$$\delta P_{upd} = Ker(C_{satisfied})\delta\alpha\,, \tag{4.3.14}$$

where $\delta\alpha$ corresponds to the coefficients in the linear combination of null space vectors. This can be further developed as:

$$\delta P_{upd} = -Ker(C_{satisfied})\left[C_{violated}Ker(C_{satisfied})\right]^{+}\delta G_{violated}\,, \tag{4.3.15}$$

$$P_{recovered} = P_{violated} + \delta P_{upd} \,. \tag{4.3.16}$$

Here, superscript $^+$ corresponds to a pseudoinverse of a rectangular matrix and usually only few Newton steps (4.3.16) are needed to recover constraints.

The implications of this approach go beyond shape optimisation and could be applied directly on CAD shapes, which do not satisfy certain requirements. To illustrate that, the Stator blade model with TE radius $r = 0.7$ mm was created using OCCT parametrisation (Fig. 4.12, grey surface). A minimum radius constraint of $r = 1$ mm was imposed at the TE which led to constraint violations at every test-point located on the edge. Results of the constraint recovery using a single Newton step are shown in Fig. 4.12, with all constraints satisfied for the updated red surface.

Figure 4.12: Recovery/Increase of TE radius from initial (grey, $r = 0.7$ mm) to updated (red, $r = 1$ mm).

4.4 Geometric optimisation with explicit and implicit parametrisations

Shape sensitivities computed with the differentiated OCCT provide crucial information on changes in CAD parameters that improve pre-defined cost function. To compare outlined parametrisation techniques (explicit parametric CAD model and implicit NURBS-based model) and showcase the impact of constraints, the

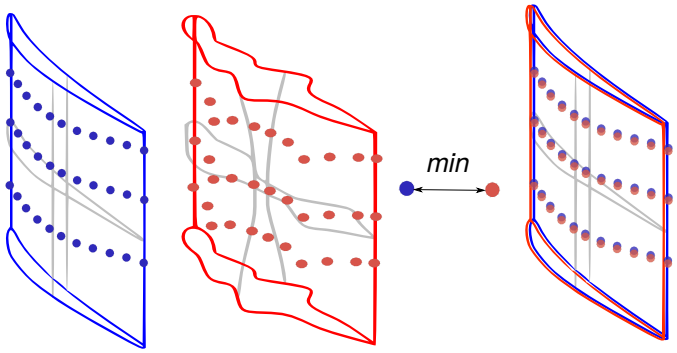

Figure 4.13: Surface points on the original (blue) and the target (red) perturbed blade.

geometric optimisation, the so-called least squares (LSQ) fitting, is performed. In Chapter 6, the fitting problem is investigated further.

Here, the goal is to fit the baseline TUB Stator blade to the intentionally perturbed (target) blade geometry (Fig. 4.13). This is achieved by the minimisation of the LSQ cost function:

$$J(P) = \|X_S(P) - T_p\|_2^2 \,, \tag{4.4.1}$$

where P are blade's CAD parameters, $X_S(P)$ corresponds to the points uniformly distributed on the original blade and T_p denotes their orthogonal projections on the target shape. Therefore, the LSQ function measures the distance between current parametrisation and the target point cloud. Contrary to the target blade and its surface points T_p, the parameters P are subject to the geometric constraints outlined in the previous sections.

1) TUB Stator blade reduction

In the first case, a perturbed target stator blade is created using parametric model with low thickness parameter values. As a result, the target blade violates the minimum thickness and TE/LE minimum radii constraints (Fig. 4.14). A constrained optimisation is performed to fit the original 3-D blade to the target. Since the target blade does not satisfy the constraints, i.e. is infeasible, an optimal feasible

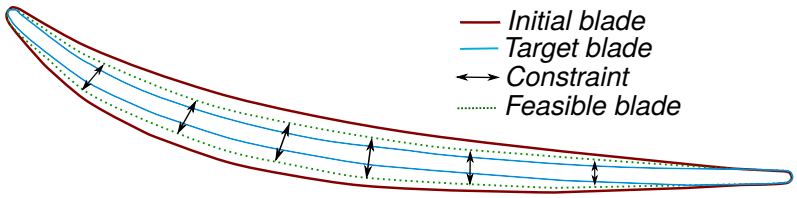

Figure 4.14: Mid-span profile for the TUB Stator blade reduction case.

blade will not match the target perfectly.

Three distinct parametrisation-optimiser settings in the Shape Optimisation Framework are used: (a) parametric (explicit) model with L-BFGS-B optimiser; NURBS-based (implicit) parametrisation with (b) Projected Gradient Descent and (c) SLSQP optimisers. The explicit parametrisation uses 184 design parameters, range values are provided to the L-BFGS-B optimiser to constrain the parameters (minimum radii, thickness). NURBS-based parametrisations consist of two surfaces with 9×22 control points in blade's axial and span-wise directions, respectively. The last and the first rows of control points (hub, shroud) are fixed, resulting in a design space with $(2 \times 9 \times 20 \times 3 = 1080)$ design parameters. Individual constraints are defined along the pairs of 18 parametric curves with 20 test-points each (Fig. 4.11). The constraints matrix is provided to the optimisers. All derivatives are computed by the differentiated OCCT in the forward vector and block-vector mode of AD for the parametric and NURBS-based parametrisations, respectively.

As expected, all three optimisations do not result in a perfect match to the target blade (Fig. 4.16) but produce the shapes that resemble the 'feasible blade' shown in Fig. 4.14 (effect of the constraints).

Two quasi-Newton optimisers from (a) L-BFGS-B and (c) SLQSP, although with different parametrisations, converge in a similar fashion (Fig. 4.15). The first-order projected gradient optimiser used in (b) is slower due to the fixed step size, but also converges to the same values as SLSQP. Line-search strategies could be implemented to accelerate the optimisation process (b).

The reason that the NURBS-based implicit parametrisation used in (b), (c) con-

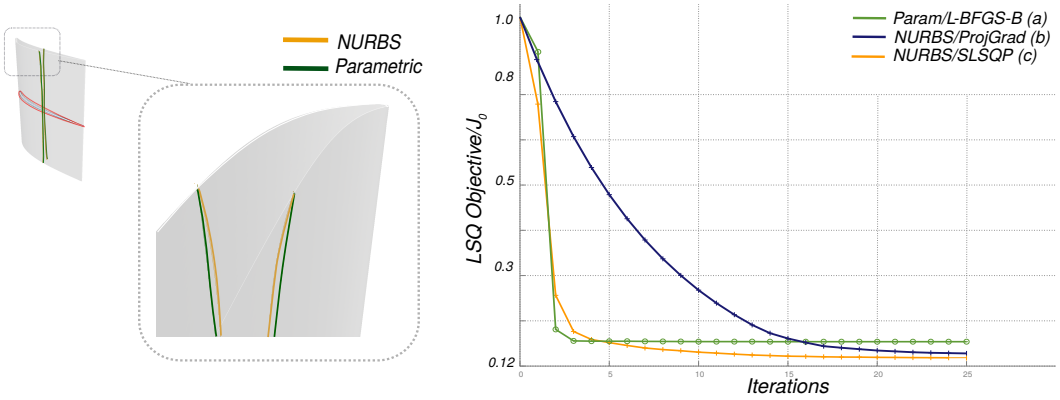

Figure 4.15: Results of the TUB blade reduction case for constrained LSQ fitting: NURBS parametrisations (b), (c) surpass parametric model (a).

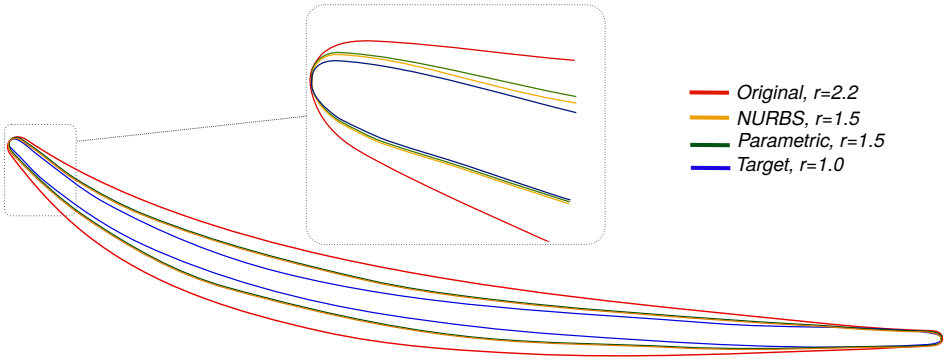

Figure 4.16: Mid-span profiles of the fitted (optimised) blades and the impact of the constraints on the leading edge radius. Parametric and NURBS-based model reach the minimum radius constraint $r = 1.5$ mm, which prevents complete fit with target.

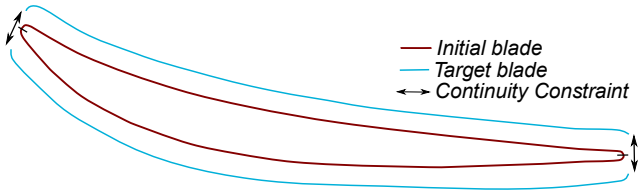

Figure 4.17: Mid-span profile for the TUB Stator expansion case.

verges to a lower level than the parametric approach is two-fold. Firstly, the NURBS space offers richer design space and more local control. Secondly, the test-points approach imposes constraints directly on the NURBS surfaces rather than on higher-level control parameters. This results in the better fitting, evident from the blade profiles in Figures 4.15 and 4.16.

2) TUB Stator blade expansion

Opposite to the previous case, here the target 3-D shape is created by an expansion of the original TUB Stator blade: its two surfaces are moved independently in the opposite outward directions (Fig. 4.17). As a result, the target blade does not satisfy continuity and min/max curvature constraints at the LE/TE. The LSQ fitting is performed using only the implicit NURBS-based parametrisation with SLSQP optimiser. The same set-up (c) as in the previous subsection is used. The unconstrained optimisation naturally results in the expansion of the blade with the continuity violation between the two surfaces along the leading and the trailing edge (Fig. 4.18).

In the first constrained optimisation, G0/G1 continuity constraints are imposed at the LE/TE. The power of the NSPCC/test-points approach is visible in Fig. 4.19, where the fitted blade maintains continuity along the edges. The G1 continuity is achieved by placing NURBS control points near the edges onto the same plane. In this case, a zero curvature is obtained at the LE/TE (Fig. 4.19, left). Another optimisation, with additional minimum curvature (maximum radius $r = 2.5$ mm) constraint at the edges, produces the rounded edge, as seen in Fig. 4.19 (right).

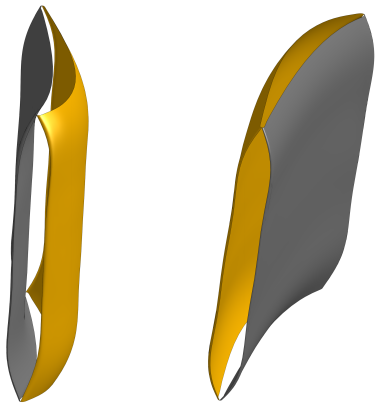

Figure 4.18: Results of unconstrained NURBS-based optimisation: continuity violation at the LE/TE.

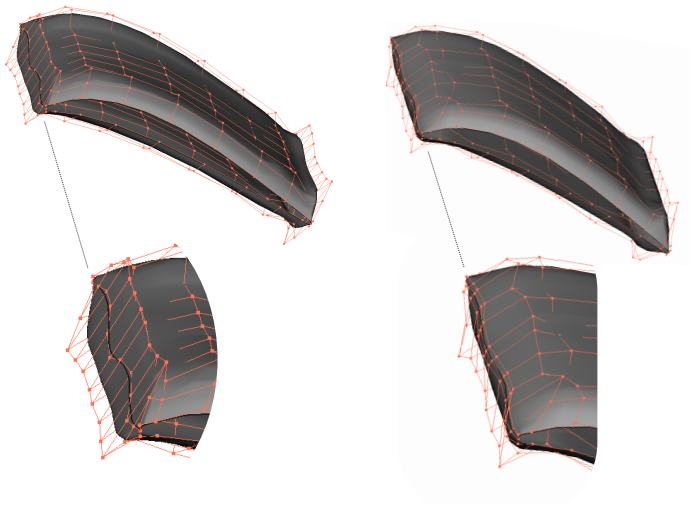

Figure 4.19: Influence of G0/G1 continuity (left) and additional minimum curvature (right) constraints on the result of LSQ fitting for TUB expansion.

4.5 Summary

In this chapter, two different possibilities for parametrisation in OCCT were showcased. Explicit parametrisations require additional time for the set-up but can capture design intent, encapsulate certain features directly into the geometrical model (e.g. constraints) and provide intuitive design space for optimisation. Of-

ten, best practices for parametrisation already exist (e.g. turbomachinery blades, aerofoils) and the use of the explicit models becomes a reasonable and convenient choice for the parametrisation. As shown here with the cross-sectional construction of the TUB Stator blade, a well-defined explicit parametrisation can produce shape optimisation results comparable to the NURBS-based models.

On the other hand, implicit parametrisations can automatically provide rich design spaces derived from NURBS, which can be refined further. The underlying NURBS space composes the intermediate parametrisation between the parametric and mesh-based approaches. Implicit models are quick to use and can be applied to both conventional and non-standard shape optimisation tasks. This work adds new capabilities to impose the surface curvature and distance constraints to the NSPCC (test-points) algorithm. The techniques to quickly generate the test-points on the surfaces, store them in the standard CAD files and visualise the constraints were proposed. These are important steps towards the adoption of the implicit parametrisation method in industrial applications.

One finds that the OCCT shape sensitivities of the TUB Stator blade are in agreement with the finite differences for both parametrisation approaches. The results also show the robustness and efficiency of the differentiated OCCT algorithms, as the sensitivities are free of the numerical errors and can be computed with the efficient modes of AD. The strength of both parametrisations and the corresponding constraints definition techniques were demonstrated by the application of several gradient-based optimisers in geometric shape optimisation (CAD fitting).

Chapter 5

Aerodynamic Optimisation of TUB Stator with Hybrid Parametrisation

This chapter presents aerodynamic shape optimisation of TUB TurboLab Stator test case. To minimise total pressure losses in the stator, two optimisation problems are set up. Firstly, the stator's blade is optimised using a conventional parametric CAD model. Secondly, the stator's hub is altered using a suitable NURBS-based parametrisation for endwall contouring. The combination of the explicit and the implicit CAD parametrisations allows achieving a significant reduction of the cost function and obtaining the optimal shape directly in the CAD format.

5.1 Stator blade optimisation

In this section, the optimisation of total pressure losses in the TUB Stator is conducted by a modification of the stator's blade. In Chapter 4, two different blade parametrisation strategies for the optimisation (parametric CAD model and NURBS-based model) were proposed. Since in this particular case both approaches provide similarly efficient design spaces (Sec. 4.4), the first parametrisation ap-

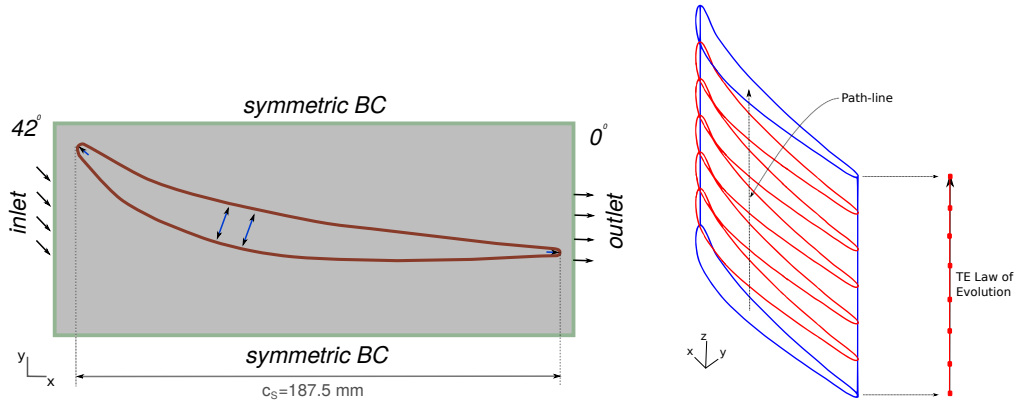

Figure 5.1: CFD simulation set-up (2-D profile view), boundary conditions (BC), manufacturing constraints and CAD model parametrisation with cross-sections.

proach is considered. Therefore, the blade's design space consists of intuitive explicit design parameters such as leading/trailing edge (LE/TE) radii, 2-D profile thickness distribution and 3-D laws of evolution (Fig. 5.1). All manufacturing constraints (minimum blade thickness, LE/TE radii, constant chord length), except the constraints for the mounting bolts, are embedded in the parametrisation together with the corresponding boundary (range) values for the parameters. Before aerodynamic shape optimisation, OCCT's blade parametrisation was also fitted to the original geometry (STEP file provided by the workshop) using least squares distance minimisation, as in Section 4.4. To recall the details of the parametrisation, the reader is referred to Section 4.2.

5.1.1 Baseline flow simulations

The STAMPS solver, recently also validated for several turbomachinery test cases [86, 88], is used for the flow simulations. The baseline flow conditions are outlined in Fig. 5.1: incoming flow with the 42° whirl angle is redirected by the blade into the axial direction. Although the TUB Stator consists of 15 blades, only a single blade is simulated and optimised using rotational periodicity. At the inlet, profile conditions for the swirl angle, total pressure and temperature are imposed. A constant mass flow $\dot{m} \approx 9.5$ kg/s is achieved by adjusting back-pressure at the outlet. The

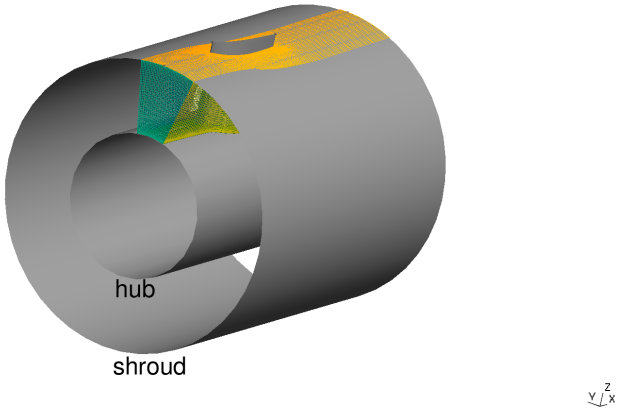

Figure 5.2: Computational domain.

total pressure loss cost function is computed by STAMPS as follows:

$$J = P_{total}|_{inlet} - P_{total}|_{outlet} \,, \tag{5.1.1}$$

$$P_{total}|_S = \frac{\int_S p_{total}\rho(u \cdot \hat{n})dS}{\dot{m}} \,, \tag{5.1.2}$$

$$\dot{m} = \int_S \rho(u \cdot \hat{n})dS \,. \tag{5.1.3}$$

Here, ρ, p_{total}, and u correspond to the fluid's density, total pressure and velocity, respectively. The cost function is computed as mass flow averaged quantity to preserve essential features of the non-uniform flow, as suggested for the turbomachinery applications in [35].

The CAD model of the blade and the cylindrical casing (hub/shroud) are used to generate a multi-block structured mesh using GridPro software [4]. The mesh is refined in the vicinity of the blade and is shown in Fig. 5.2. The sequence of the generated meshes with 400k, 800k and 1.9M cells is later used in the grid convergence study. The dimensionless wall distance values y^+ for the meshes are 70, 40 and 3, respectively. Although for the coarser meshes the high-fidelity of the near-wall flow solution is not guaranteed, the grids can adequately capture important flow features that influence the cost function (e.g. flow separation at the blade's wake).

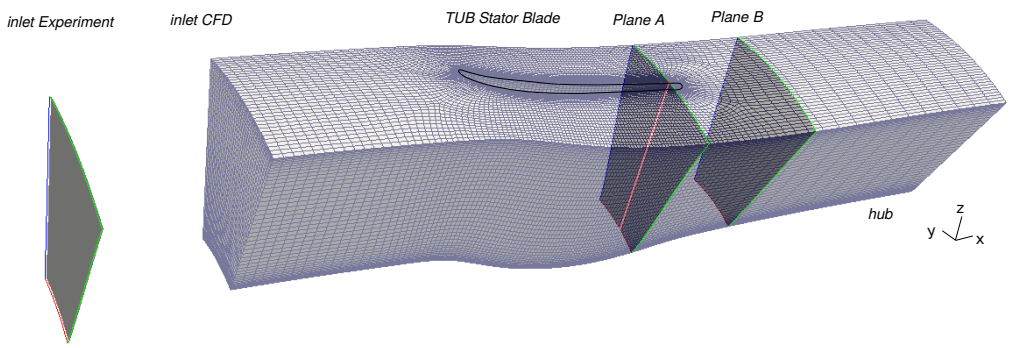

Figure 5.3: 400k CFD mesh and the measurement planes.

In the scope of About Flow benchmarking workshop [12], several research groups from academia and industry have worked on the TUB Stator optimisation [119]. In particular, numerical and experimental investigation of the baseline and the optimised blade was conducted by the researchers at Rolls-Royce Deutschland and TU Berlin (Mihalyovics et al. [85]).

In comparison to the flow conditions specified by the workshop and used here for the CFD simulations, the experiment involved slight modifications (Fig. 5.3). The experimental set-up imposed inlet profile conditions at a distance of $3c_s$ upstream of the leading edge of the blade, contrary to the $1.0c_s$ for the CFD simulations. Here, $c_s = 187.5$ mm is the blade's chord length. Despite the discrepancy in the inlet boundary condition, the flow simulated by the solver still can be compared with the experimental results. Measurement planes A and B (Fig. 5.3) are located at the trailing edge and $0.6c_S$ downstream, respectively. Figure 5.4 shows velocity profiles for the sequence of meshes at the plane B with the corresponding experimental results. The numerical results on the fine mesh (1.9M) are in fair agreement with the experimental data and can predict low velocity (hub corner separation) in the wake of the blade (Fig. 5.4). Similar comparison results were also reported by Vasilopoulos et al. [85]. The mid-size mesh (800k) also captures the hub separation and is qualitatively similar to the experiment. On the other hand, the coarsest mesh predicts also a significant shroud vortex, not present in the experimental results. The mid-size mesh is therefore chosen for the optimisation as a good compromise

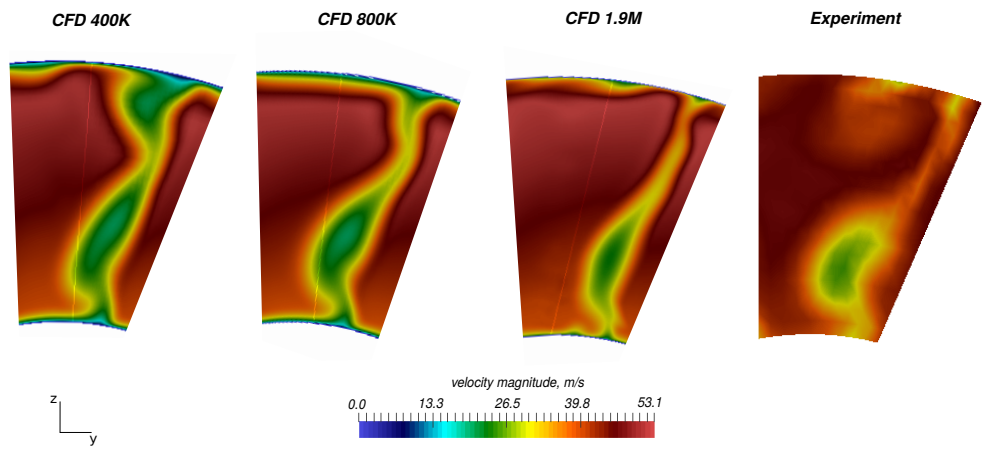

Figure 5.4: Velocity magnitude at the wake of the blade by STAMPS (mesh sequence 400k/800k/1.9M cells) and Experimental data at Plane B.

between flow fidelity and the simulation time.

Figure 5.5 shows the adjoint CFD sensitivity field computed by STAMPS for the baseline geometry. The cost function sensitivity w.r.t. movement of the mesh nodes in the direction normal to the blade and the hub surfaces is demonstrated. The non-smooth sensitivity field has the highest absolute values on the blade's suction side in the areas affecting the flow separation. The shape modification in these regions can therefore effectively reduce J.

5.1.2 Optimisation

The Shape Optimisation Framework is used to minimise the pressure loss cost function J using L-BFGS-B optimiser. At every iteration, STAMPS calculates CFD sensitivity: primal and adjoint solver runs are terminated when a decent level in residual convergence of state equations is achieved ($7 - 8$ orders of magnitude). Both primal and adjoint computations in STAMPS are performed in parallel on 32 cores. The differentiated OCCT computes parametric CAD sensitivity in the reverse mode of AD, and therefore the total gradient is obtained by fully differentiated (reverse) design chain. Afterwards, the CAD model and the computational mesh are updated accordingly. At each new optimisation step, the flow field is

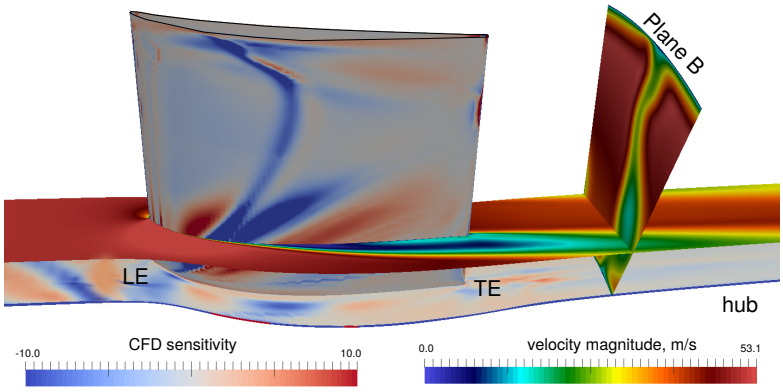

Figure 5.5: Blade and hub 'push/pull' (blue/red) CFD sensitivity; velocity magnitude along the stator and plane B.

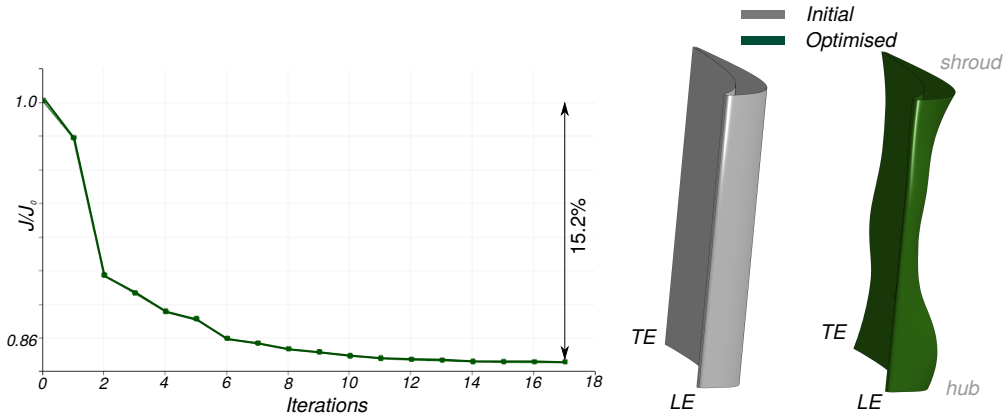

Figure 5.6: Optimisation history, initial and optimised stator blade.

re-initialised with the solution from the previous iteration. For moderate shape changes, this accelerates the simulation process.

After seventeen iterations, the optimiser improves the objective function by more than 15% (Fig. 5.6). The parametric design space allowed considerable shape modifications: the overall thinner blade with significant camber line displacement at the mid-span and a leaned trailing edge is obtained (Fig. 5.6, Fig. 5.7). A similar optimal shape, with considerable camber line movement at the mid-span towards the pressure side, was also reported in the reference [119].

The major pressure loss improvements can be attributed to the reduction of the

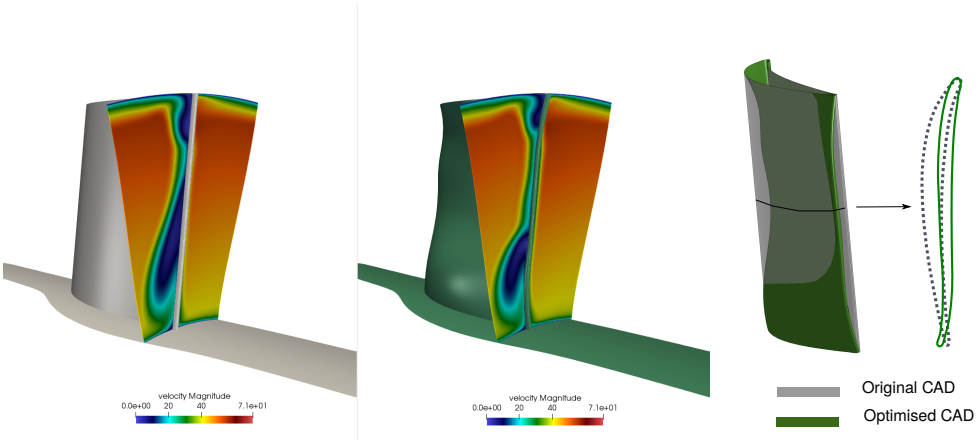

Figure 5.7: Velocity magnitude for the baseline and the optimised blade at the TE/Plane A (left); difference in CAD geometries (TE view) and mid-span profile comparison (right).

corner flow separation, as shown in Fig. 5.7. The shape modifications incur a stronger flow at the mid-span and reduce the low-velocity regions near the hub and the shroud. This also decreases flow circulation originating from the hub vortex, as shown in Fig. 5.8.

Another mechanism for the pressure losses reduction is a deviation of the flow from the axial direction at the outlet (Fig. 5.9). Usually, thinner blades have milder loading and therefore worsen the turning of the flow. Here, this was penalised by the minimum thickness constraint, which had a notable impact on the optimisation result. The unconstrained optimisation, for instance, can lead to a complete reduction of the blade thickness. The problem can be also addressed with the additional exit whirl angle cost function, specified by the TUB Stator workshop. As shown in [119], this objective function is contradicting the pressure loss reduction and leads to a different optimal shape. For this multi-objective optimisation problem, the same CAD-based approach can be applied in the future.

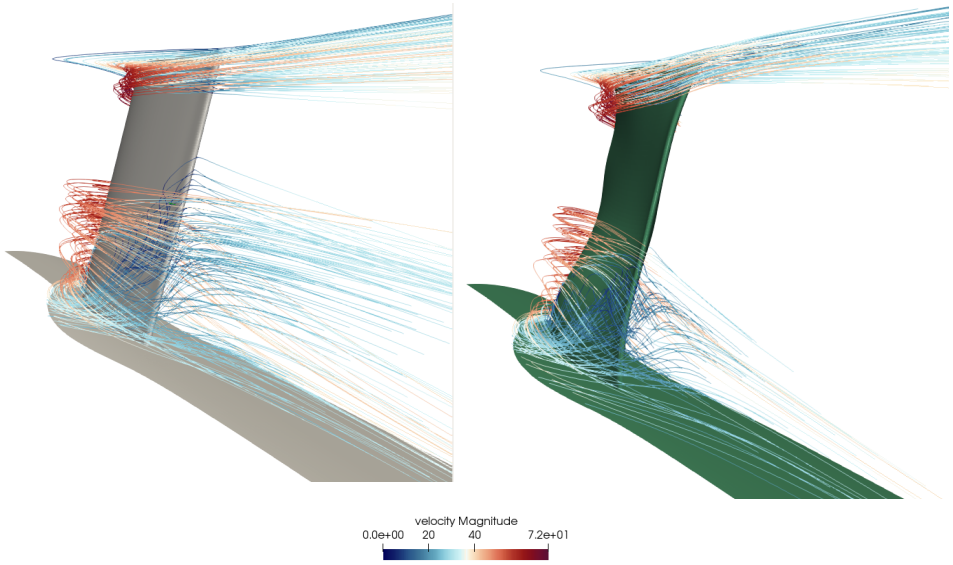

Figure 5.8: Streamlines at the trailing edge for the baseline (left) and the optimised (right) blade.

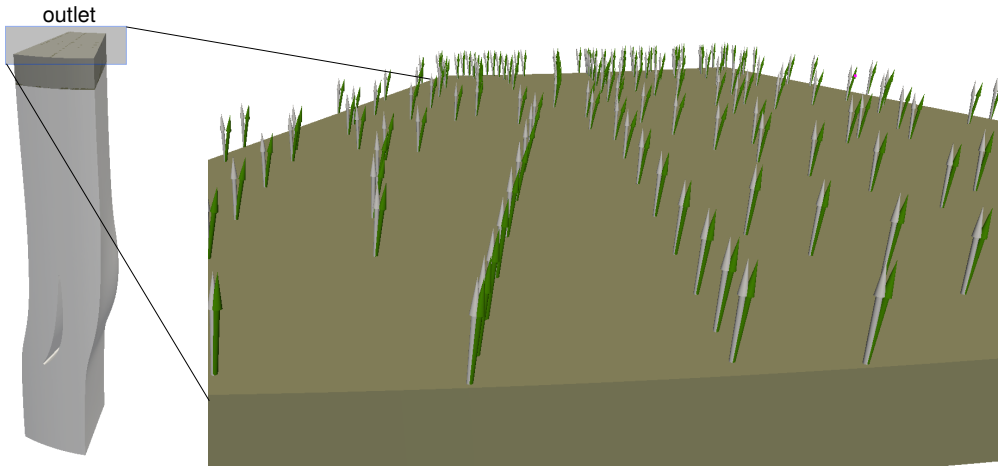

Figure 5.9: Flow exit angle deviation from the axial (normal) direction at the outlet: baseline (grey) and optimised for pressure losses (green).

5.2 Stator hub endwall design

In addition to the shape of the blade, the test case also allows modifications of the stator's hub (endwall contouring). The baseline geometry of the hub is provided in the form of a canonical cylindrical surface which is not suitable for the contouring.

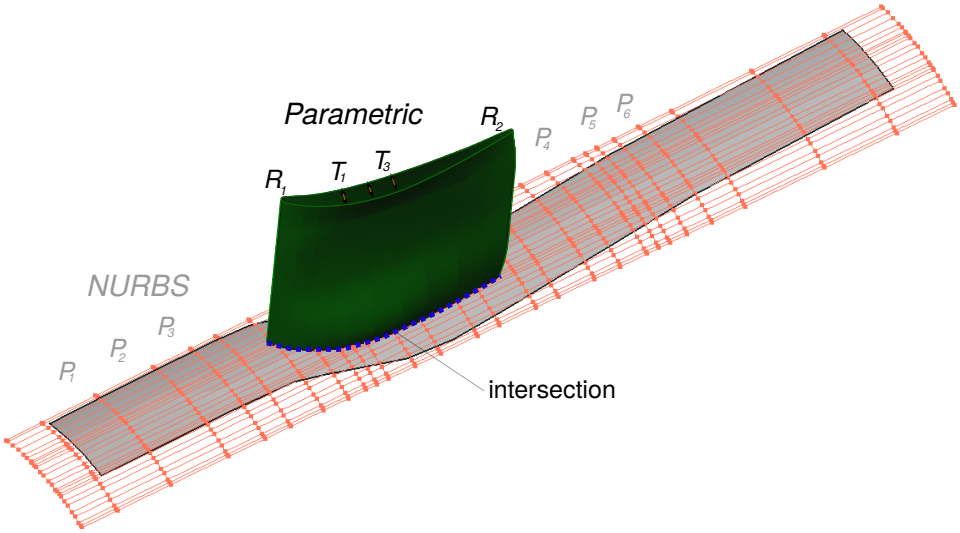

Figure 5.10: Hybrid CAD design: parametric blade model and NURBS ($l1$) hub surface.

Instead, a more useful NURBS-based parametrisation is reverse engineered from the available hub's surface grid points. OCCT and the gradient-based least-squares optimisation are used for the topology generation and CAD-to-mesh fitting procedures (example of topology generation procedure is detailed in Section 6.2). The resulting cubic NURBS surface (Fig. 5.10) has a (28×25) control point net and produces a suitable fit with the mesh (maximum mesh-CAD distance/tolerance $O(10^{-3})$ mm is consistent with the near-wall grid resolution). Geometrically, the surface is a rectangular patch outlined by the red control points in Fig. 5.10, while the topological CAD face (in grey) covers only the periodic grid surface of the hub. The discussion on the differences between CAD topology and geometry can be found in Section A.2.

Additionally, a refined NURBS parametrisation was constructed by the uniform insertion of new knots into the hub surface. The two parametrisations are referred here to as $l1$ (initial) and $l2$ (refined) with details presented in Tab. 5.1 and Fig. 5.11. Both parametrisations are later used for the endwall optimisation.

The re-parametrised surface of the hub is combined with the previously optimised blade and the same flow set-up and the optimisation goal (total pressure loss min-

Hub Parametrisation	Control Points (CP)	Design CP	Parameters	ΔJ
l1	28×25	9×25	675	0.13%
l2	46×34	12×34	1224	0.3%

Table 5.1: NURBS parametrisations of the hub.

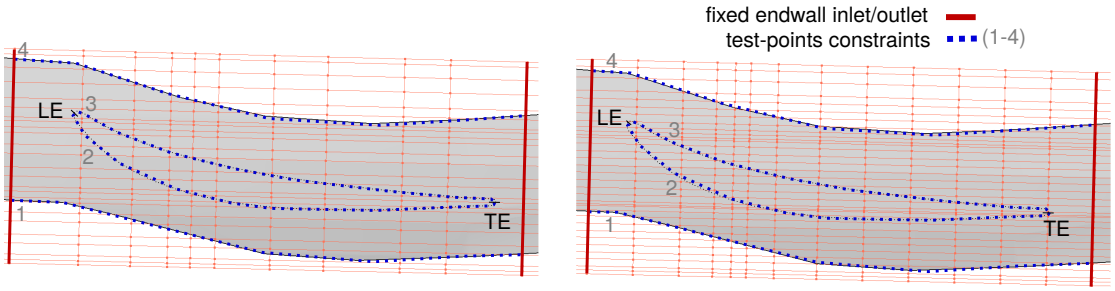

Figure 5.11: NURBS design spaces and constraints of the hub surface: $l1$ (9×25 CPs, left) and $l2$ (12×34 CPs, right).

imisation) are considered. Here, the blade remains fixed during the optimisation and only the hub's control points are modified. The movement is allowed in the neighbourhood of the blade between fixed endwall inlet/outlet, which amounts to 675 ($l1$) and 1224 ($l2$) design parameters shown in Fig. 5.11. Also, the intersection with the blade and the periodic boundaries of the hub surface are kept fixed using the test-point approach. Two hundred test-points are distributed along the four edges to impose the constraints (excessive test-points are filtered out by the NSPCC method (Sec. 4.3)). The constraints ensure that the computational domain remains watertight and periodic during the optimisation.

5.2.1 Endwall optimisation

The differentiated OCCT is used in the forward block-vector mode to compute the constraints matrix and CAD sensitivities. The Shape Optimisation Framework provides projected (constrained) gradient to the L-BFGS-B optimiser and two optimisation procedures with $l1$ and $l2$ parametrisations are conducted. After four iterations, the optimiser achieves 0.13% and 0.3% cost function improvement

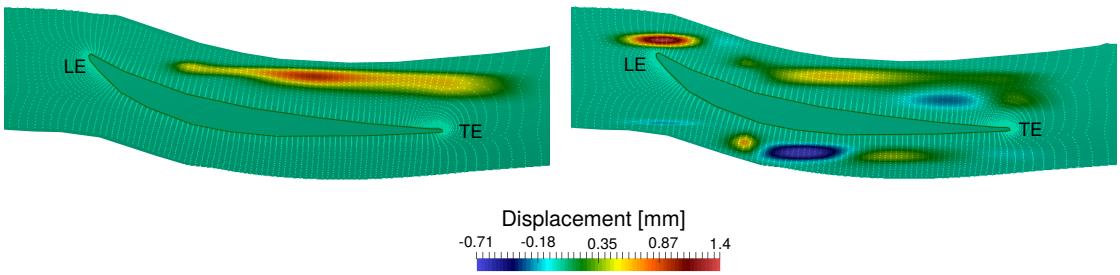

Figure 5.12: Optimised endwall contours (radial displacement) for $l1$ (left) and $l2$ (right) parametrisations (characteristic blade dimensions: height $h_s = 150$ mm, chord length $c_s = 187.5$ mm).

(ΔJ, Tab. 5.1) for the $l1$ and the $l2$ parametrisation, respectively. The corresponding hub deformations are illustrated in Fig. 5.12. Further optimisation iterations resulted in larger displacements and were not considered due to the deteriorating mesh quality. The above-mentioned cost function improvements offer additional gains to the previously optimised blade geometry (15.2%).

Here, the design space and the constraints had an important impact on the optimisation outcome, since only changes in the limited regions between fixed periodic boundaries and blade's intersection edges were allowed (Fig. 5.12).

The optimised hub with the $l1$ parametrisation has a positive radial displacement ('hill') only on the pressure side. The resulting contour resembles the shape of the blade and helps to 'guide' the flow. In turn, this reduces the flow separation at the hub and minimises the pressure losses. There are no shape changes at the suction side where the $l1$ control points movement would have caused constraints violations.

Additional degrees of freedom in the $l2$ parametrisation allowed the movement of control points on both sides of the blade. Positive and negative ('valley') displacements are captured from the baseline CFD sensitivity field (Fig. 5.5) with the distinct hill near the blade's leading edge. Naturally, the richer design space leads to a higher cost function reduction, however, it was still limited by the constraints. Chapter 7 introduces a parametrisation technique that can alleviate this drawback:

the position of the intersection line between moving and fixed surfaces (in this case the hub and the blade) can be also included into the design space.
The endwall modifications obtained here resemble results for another turbomachinery cascade configuration [104], where blade optimisation and the endwall contouring (without geometric constraints) were performed with evolutionary algorithms. In this thesis, a milder cost function improvement was obtained from the constrained surface modification (maximum endwall deformation is 1% of the blade height). Since the endwall losses stem from the viscous effects and the corresponding secondary flows at the boundary [74], in the future, higher fidelity CFD calculations on the fine mesh can be conducted to improve the endwall shape further. As shown here, the same gradient-based optimisation setup can be effectively applied in the process.

5.3 Summary

This chapter demonstrated the successful application of a fully differentiated Shape Optimisation Framework to the aerodynamic design of the TUB Stator test case. The framework, equipped with the efficient AD derivatives of CAD model parametrisation, geometric constraints and aerodynamic simulations, streamlined the process of gradient-based optimisation. OCCT allowed combining seamlessly two parametrisation techniques, namely the parametric and the NURBS-based approaches, for the optimisation of a single component. The parametric model of the stator blade was created with the conventional turbomachinery parameters and design practices, while the reverse engineering process was used to set up the automatic NURBS-based parametrisation of the stator's hub. This novel CAD-based approach with the hybrid blade-hub parametrisation facilitated a significant decrease in the total pressure losses in the stator. The optimised blade and the hub are obtained in the CAD format, which makes the stator passage readily available for further analysis (e.g. re-meshing, numerical simulations).
Future work will be focused on the simultaneous optimisation of the blade and

the hub, rather than the two optimisation procedures conducted for each part successively. A richer and more flexible design space can be obtained by including the moving blade-hub intersection line into the optimisation. The corresponding technique is demonstrated in Chapter 7. Development of periodic boundary conditions for CAD surfaces (e.g. allowing only synchronous control point movement on periodic boundaries) can remove limitations of fixed boundary constraints and further improve the design space and the optimisation results.

Chapter 6

Re-parametrisation and Optimisation of TUM DrivAer Mirror

The chapter presents shape optimisation of TUM DrivAer vehicle to reduce aeroacoustic noise generated by its side mirrors. The test case introduces several typical challenges for CAD-based optimisation of industrial components. An explicit parametric CAD model is not available, the geometry provided in the standard CAD file also does not define a suitable implicit (BRep) parametrisation for optimisation. While the generation of a parametric CAD model of TUM DrivAer mirror could require time-consuming reverse engineering effort, the re-parametrisation of the implicit model is proposed. A BRep with a new topology and geometry is designed and then fitted to the original model using the Shape Optimisation Framework. The differentiated CAD is an essential component in the fitting procedure: it enables morphing of multi-patch CAD geometries to point clouds using gradients of user-defined distance metrics. The resulting mirror parametrisation with improved design space is then used for the aerodynamic optimisation and outperforms several alternative approaches developed by IODA partners.

The aerodynamic optimisation of the mirror neck and in particular the CFD simula-

tions of the DrivAer vehicle was conducted in collaboration with Christos Kapellos (IODA Research Fellow at Volkswagen AG).

6.1 TUM DrivAer vehicle test case

The TUM DrivAer test case (Fig. 6.1) was designed to accelerate research on car aerodynamics for realistic geometries which resemble shapes of modern production vehicles [10]. The aerodynamic shape optimisation of the DrivAer case

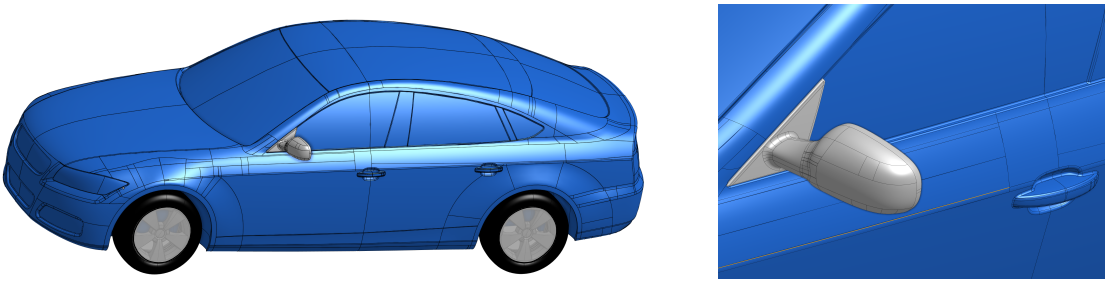

Figure 6.1: Geometry of the TUM DrivAer vehicle and its side mirror.

was proposed by the AboutFlow workshop [13]. Here, we investigate one of the aerodynamic optimisation scenarios outlined by the workshop: minimisation of an acoustic signature of the car's side mirrors. The optimisation is considered for the car travelling at the speed of 140 km/h.

Geometry

The test case subdivides the geometry of the car into several groups (body, wheels, back of the car, side mirrors, etc.). For each group, several geometrical modifications are available, which can be composed into car configurations of different type or complexity (e.g. smooth/detailed underbody, fast/estate back of the car, closed/open wheels, etc.). The test case provides multiple CAD files (STEP, IGES) for the groups described above. Here, we consider the fast-back geometry with side mirrors, smooth underbody and closed wheels, as shown in Fig. 6.1. The CAD model of the mirror consists of a large number of patches/surfaces, which poses

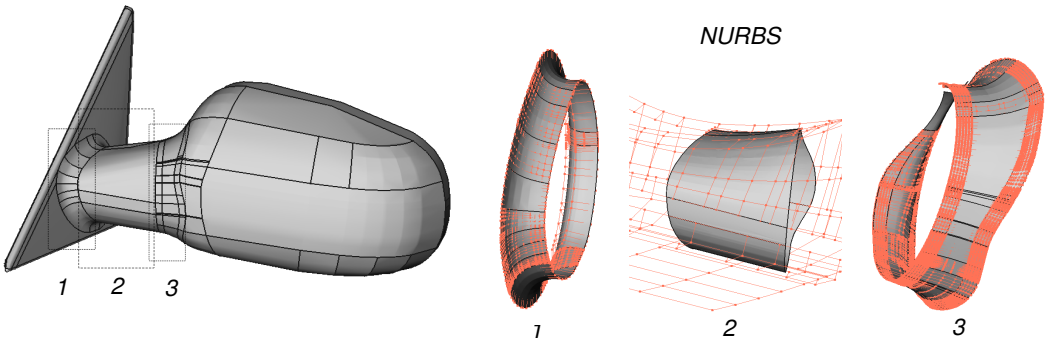

Figure 6.2: BRep of the mirror neck.

challenges for the optimisation and design space definition. In the next section, ways to handle such complex geometries and make them suitable for CAD-based optimisation are described. For this case, only geometric continuity constraints are imposed (geometry remains watertight during optimisation).

6.2 Mirror neck re-parametrisation

Often, the BRep data stored in the standard CAD files are not suitable for direct application of NURBS-based optimisation techniques. The problems stem from inefficient design spaces:

- very dense control point nets on CAD patches;
- uneven distribution of control points between the patches;
- large number of NURBS patches (several dozens), which complicates imposition of cross-patch continuity constraints;
- complex CAD patch topologies with multiple edges/faces/trims.

Generally, this could be considered as the over-parametrisation of the geometric model, since a similar shape could be obtained with a simpler design space. Figure 6.2 highlights these problems for the DrivAer mirror neck: a relatively simple component consists of several dozens of NURBS patches with non-homogeneous

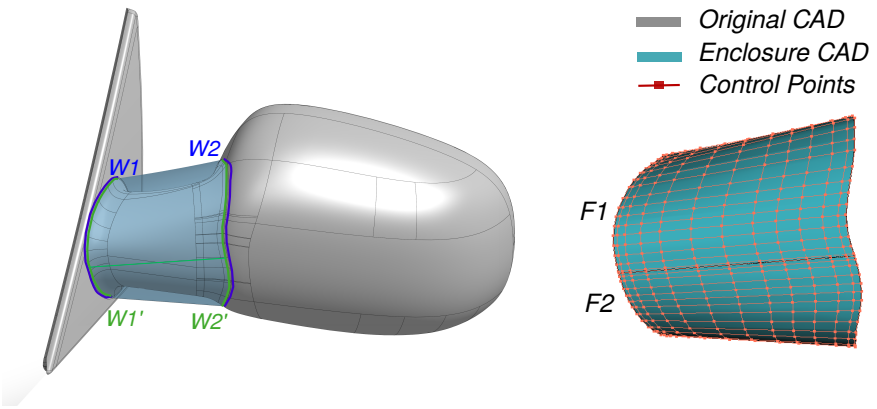

Figure 6.3: Left: Original mirror and CAD enclosure; right: BRep of the enclosure.

control point distribution. To alleviate these problems, the mirror neck is re-parametrised with the differentiated OCCT. In the two-stage process, the new CAD topology is generated and then morphed to match the original geometry. Firstly, the edges on both sides of the neck are determined with OCCT and concatenated to two wires $W1$ and $W2$ (Fig. 6.3). In OCCT terminology, the wire is a collection of edges, where edge pairs have a common vertex. In the vicinity of $W1$ and $W2$, another two circular wires $(W1', W2')$ are constructed, each consisting of two edges. Afterwards, the four wires are provided to the OCCT lofting algorithm (`BRepOffsetAPI_ThruSections`, Listing A.1), which generates the cylindrical enclosure around the mirror neck (Fig. 6.3). The topology of the wires determined the result of the lofting operation. The two-edged wires $W1', W2'$ contributed into the simple two-faced CAD enclosure $(W1' - W2')$. The short lofted surfaces $(W1 - W1')$ and $(W2' - W2)$ ensured smooth transition between the enclosure and topologically complex adjacent patches on the mirror body. The CAD enclosure faces $F1$ and $F2$, which are designed to substitute the mirror neck, also were subject to knot insertion procedure. This allowed improving the uniformity of the control point distribution, compared to the surfaces obtained directly from the lofting algorithm. Figure 6.3 shows the resulting cubic NURBS surfaces with 12×25 (F1), 12×29 (F2) control points net. Here, the knot insertion allowed the generation of a rich and flexible design space with 1944 design parameters ($n = 12 \times 54 \times 3$).

6.2.1 Differentiated CAD for fitting

In the second stage, the Shape Optimisation Framework is used to fit the new CAD enclosure to the original neck. A surface mesh is generated on the original geometry and then projected onto the enclosure. The least squares (LSQ) distance J between the target mesh and the projected points is minimised with the *L-BFGS-B* optimisation algorithm:

$$\min_P J(P) = \frac{1}{m} \sum_{i=1}^{m} \|X_i(P) - T_{p_i}\|_2 \,,$$

$$P^{(k+1)} = P^{(k)} - \mathcal{A}(t_k, \nabla J(P^{(k)})) \,.$$

Here, P denotes the control points of the CAD enclosure, T_p - the target mesh points of the original neck, X - their orthogonal projections on the enclosure and $\mathcal{A}$ - the gradient-based optimisation algorithm and its iteration step size t. The cost function's gradients are computed in OCCT with the efficient block-vector mode, which enables the use of the rich design space (1944 parameters). CAD sensitivities w.r.t. 120 parameters were computed with the vector mode of ADOL-C for each of 16 consecutive blocks, with the remaining parameters in a smaller block. The size of the block (120) was chosen to limit the memory footprint of the differentiated OCCT.

Since the re-parametrised mirror neck consists of two independent patches $F1$ and $F2$, the continuity between them might be violated during the optimisation. The framework allows avoiding this using the test-points/NSPCC approach. Forty test-points are distributed on the cross-patch edges to impose G0 continuity. The amount of the test-points is set before the fitting procedure based on the surface properties of $F1, F2$ (degree/control point distribution) and the rank of the constraints matrix, as explained in Subsection 4.3.2. Differentiated CAD computes the constraints matrix and assembles the projected (constrained) gradient, which is later supplied to the optimiser.

Figure 6.4 shows the target mesh with $m \approx 8000$ surface points, the initial and the fitted CAD enclosure. The LSQ distance, although with a few 'jumps' caused by

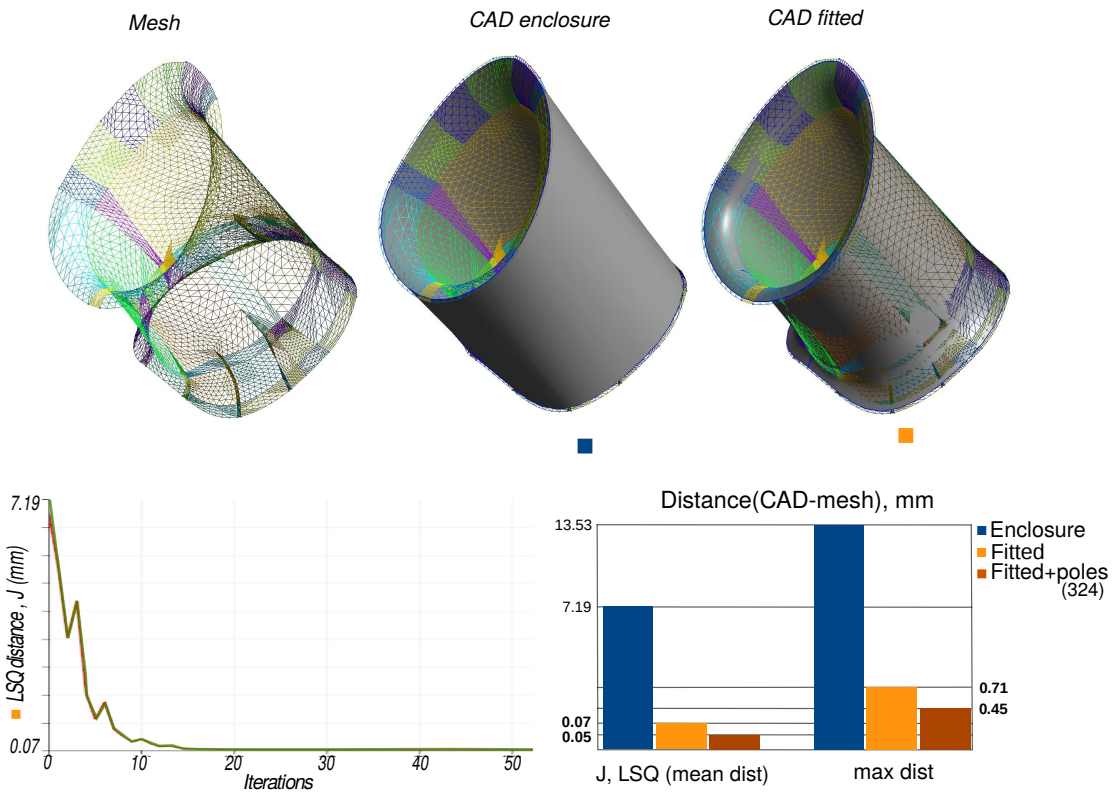

Figure 6.4: CAD-to-mesh fitting and CAD-to-point distances.

the L-BFGS-B step sizes in the initial stage of optimisation, is reduced significantly and results in a decent fit between the re-parametrised and the original neck. While a characteristic dimension of the neck (width, height) is approximately 100 mm, the LSQ fitting achieves an adequately small distance ($J^* = 0.07$ mm) between the fitted CAD surfaces and the target mesh points. This is sufficiently smaller than the cell sizes of a CFD grid (1.5 mm), which is used for aerodynamic simulations in the next section.

It is important to note, that the LSQ distance measures the mean discrepancy between two point sets. As a result, the quality of the fit in certain regions might be higher than the others. As shown in Fig. 6.4, the maximum distance between the target mesh points and the fitted CAD geometry (0.71 mm), although is acceptable in this case, is naturally greater than the LSQ value. While this is not critical for the neck re-parametrisation, the problem can be solved with alternative

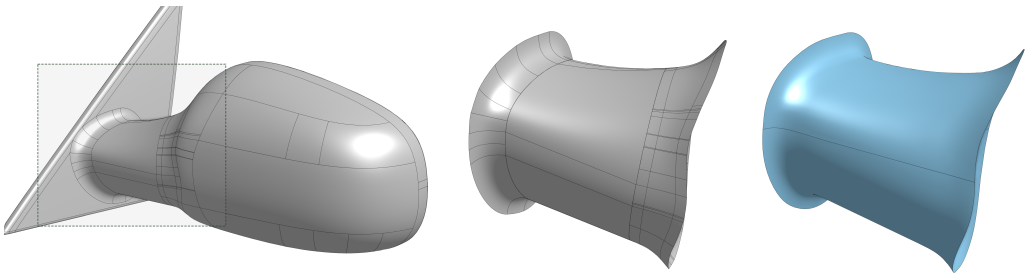

Figure 6.5: Original mirror and neck (grey) and fitted enclosure (blue).

distance functions or by further design space refinement. The CAD enclosure with additional 324 control points (obtained with knot insertion), located on the sides of the mirror, allows achieving better fitting in these areas and improves both the LSQ and maximum distance metrics (Fig. 6.4).

The differentiated OCCT can be used to obtain gradients of other geometric functions. Several differentiable distance metrics such as Chamfer or Hausdorff functions [43], although at higher computational cost than LSQ, penalise stronger discrepancies between individual point pairs and could be considered. In a similar fashion, OCCT can be used to incorporate the point's normals and curvature information into a distance metric, which can further improve the high-fidelity CAD fitting procedure. The flexibility in the choice of a design space and distance functions makes OCCT a useful, practical tool for the reverse engineering and CAD fitting tasks (e.g. point clouds to CAD), which are often occurring in industrial workflows.

Finally, the neck part in the original CAD model is substituted with the re-parametrised shape (enclosure), which makes the design space of the DrivAer mirror neck well-suited for the aerodynamic optimisation (Fig. 6.5).

6.3 Aerodynamic optimisation

6.3.1 Continuous adjoint formulation

The section investigates minimisation of noise generated by the TUM DrivAer mirror. While mirror design presents an important and challenging aeroacoustic problem, several works proposed the use of a surrogate model to measure the car's interior noise [18, 98]. Since it can be correlated with the level of the turbulence outside the vehicle, the authors formulate a corresponding cost function as the volume integral of the squared turbulent viscosity. While this is a decent surrogate model, particularly for the low-frequency noise, in the future, higher fidelity aeroacoustic models can be considered with the same optimisation approach. For the sake of complicity, the surrogate model used for the DrivAer mirror optimisation is briefly outlined:

$$J_{noise} = \int_{\Omega} \nu_t^2 d\Omega \,, \tag{6.3.1}$$

subject to steady RANS equations

$$R^p = \frac{\partial u_j}{\partial x_j} = 0 \,, \tag{6.3.2}$$

$$R_i^u = u_j \frac{\partial u_i}{\partial x_j} - \frac{\partial \tau_{ij}}{\partial x_j} + \frac{\partial p}{\partial x_i} = 0 \,. \tag{6.3.3}$$

Here, the volume Ω is an extrusion of the DrivAer side window by 3 cm (Fig. 6.6), the state variables (u, p) correspond to velocity and the static pressure normalised by the fluid density. Repeated indices should be considered as a summation. The stress tensor

$$\tau_{ij} = (\nu + \nu_t)(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i})$$

includes kinematic viscosity ν and turbulent viscosity ν_t, also present in the cost function. The objective is to change the shape of the DrivAer mirror neck in order to minimise J_{noise}.

E.M. Papoutsis-Kiachagias et al. [98] use a continuous adjoint formulation to derive adjoint equations for the system (6.3.2)-(6.3.3), corresponding boundary conditions

Figure 6.6: Objective function is calculated over the volume Ω (red).

and the model's sensitivity. Since the cost function J_{noise} depends on the turbulent viscosity, the authors also differentiate the Spalart-Allmaras turbulence model. For further details of the approach and the verification of the model, the reader may refer to the publications [98, 99].

6.3.2 Optimisation

For the aerodynamic simulations, the OpenFOAM-based CFD solver with primal and adjoint capabilities is used. The solver is maintained by Volkswagen and was extensively tested in the industrial setting for applications in automotive aerodynamics [72]. In a separate study, the numerical solution computed by the OpenFOAM solver for the TUM DrivAer model also shows good agreement with the experimental wind tunnel results [56].

The steady-state flow equations are solved with standard incompressible SIMPLE-type algorithm. The code is augmented by the corresponding differentiated adjoint routines that can compute CFD sensitivity of the surrogate aeroacoustic model, as outlined in the previous section. The solver also implements mesh morphing using Laplacian smoothing.

The mirror with the re-parametrised neck is combined with the full TUM DrivAer vehicle CAD model, which is used for the CFD mesh generation. The OpenFOAM

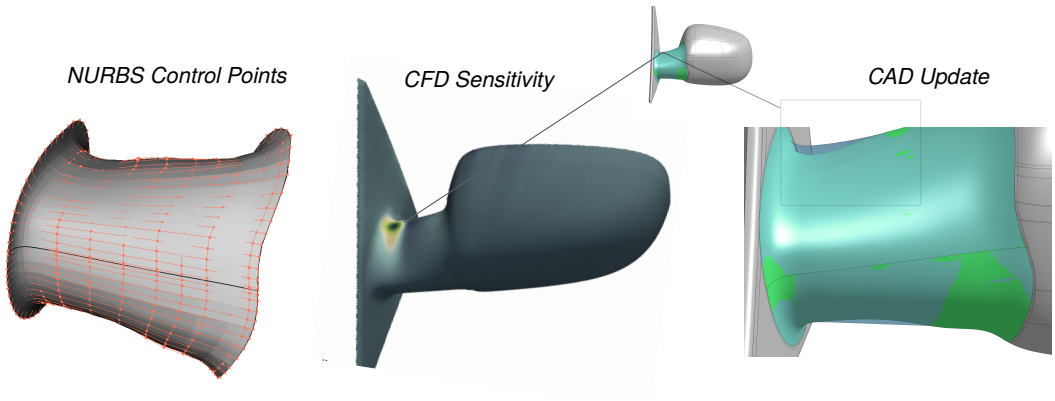

Figure 6.7: Left: NURBS space of the re-parametrised neck; middle: initial CFD sensitivity; right: original (blue) and aerodynamically optimised (green) CAD shape.

case is set up to produce the unstructured mesh with the `snappyHexMesh` executable. The mesh is refined in the vicinity of the mirror to ensure the accuracy of turbulent viscosity calculation. In total, more than five millions mesh cells are generated. The prescribed boundary conditions correspond to the freestream velocity of 140 km/h.

Finally, the results of the CFD simulations - the cost function value and its gradients are provided to the Shape Optimisation Framework for CAD-based mirror neck optimisation. The same CAD set-up and design space as in the re-parametrisation problem is considered (NURBS control points). First rows of control points on the sides of the neck remain fixed to ensure $G0$ continuity of the neck with the rest of the mirror. Forty test-points are distributed along the two edges of the neck to impose cross-patch $G0$ continuity constraints (faces $F1, F2$). Together with the CFD and CAD sensitivities, the constraints are supplied to the projected gradient descent optimiser, available in the framework. At each optimisation iteration, the NURBS control points and hence the shape of the mirror neck are updated. The changes in the CAD shape are also propagated into the interior mesh nodes by solving the Laplace equation (mesh smoothing), so no re-meshing is performed during the optimisation.

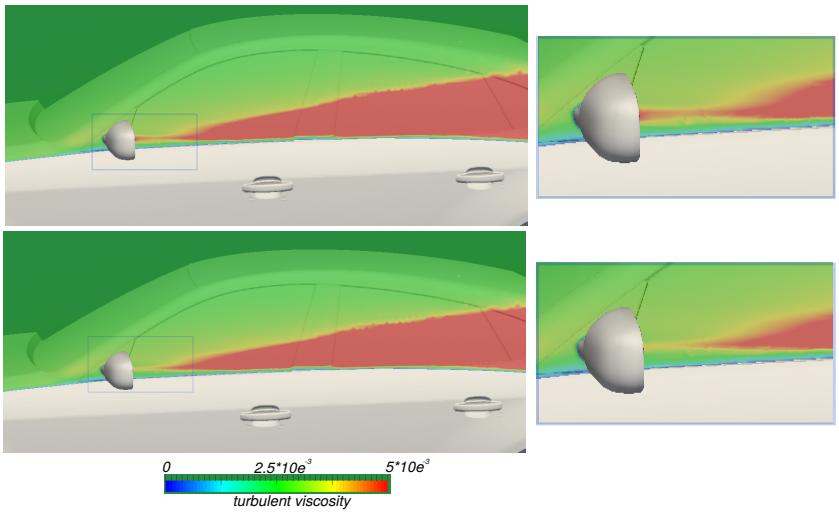

Figure 6.8: Turbulent viscosity ν_t at the side of the car for baseline (top) and optimised (bottom) mirror neck.

After seven optimisation iterations, the initial 'noise' cost function is reduced by 15.12%. Figure 6.7 demonstrates the optimised geometry of the mirror. The initial CFD sensitivity field has high 'push surface in' values located on the mirror neck, in the vicinity of the domain Ω. The design space of the optimised neck allowed obtaining the major CAD model update in this area. As a result of this shape deformation, a strong decrease of the turbulent viscosity downstream from the mirror is observed (Fig. 6.8), which in turn reduces the 'noise' objective. While here the single-objective aerodynamic optimisation problem was considered, without limitation, the same framework can be applied to address further objective functions or constraints (e.g. aerodynamic drag constraint). The additional CFD cost function and its sensitivity can be provided to the available non-linear optimisers.

6.3.3 Comparison with alternative parametrisations

Additionally to the OCCT-based DrivAer mirror optimisation, IODA's project partners conducted three other optimisation procedures. The analogous flow set-up was used with different parametrisation techniques. The achieved optimisation results are provided in Table 6.1. The comparison with the first two mesh-based

A. Mesh Smoothing	B. Vol. B-splines	C. Parametric CAD/FD	D. OCCT AD
11.94 %	5.72 %	5.14 %	15.12 %

Table 6.1: Surrogate objective function optimisation results with different parametrisation techniques.

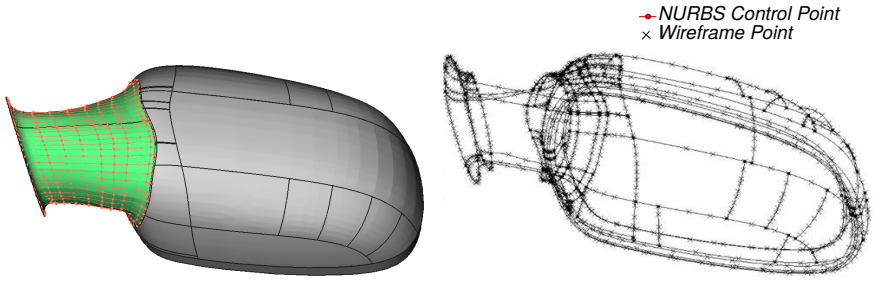

Figure 6.9: NURBS (neck) and wireframe [18] (full mirror) design space.

approaches (A. Implicit mesh smoothing [21], B. Volumetric B-splines [98]) is omitted here, instead, the differences between the relevant CAD-based parametrisation techniques (C [18] and D(OCCT AD)) are highlighted.

In the approach C, Agarwal et al. [18] use their shape optimisation framework based on the commercial CAD system CATIA V5 and its functionality to compute sensitivities of the parametric CAD models with finite differences (FD). The DrivAer mirror was firstly re-parametrised: the authors do not build a traditional parametric model (e.g. section profiles/3-D operations) but generate a wireframe object using characteristic points of the original mirror STEP file (edges, vertexes, etc.), as seen in Fig. 6.9. Afterwards, the wireframe with 3048 points-parameters is used to generate the external surfaces. Since the computation of all CAD model derivatives with FD is computationally expensive, the authors determine 7 most effective parameters, which are later used to drive aerodynamic optimisation.

For this particular test case, the advantage of the OCCT approach D is two-fold. Firstly, the shape fitting allowed us to discard original inefficient STEP topology and to substitute it with the flexible design space. The resulting NURBS control points were well-suited to accommodate the features of the CFD sensitivity

field. Secondly, the computation of CAD derivatives with the efficient block-vector AD mode allowed us to obtain the sensitivities in a couple of minutes and did not penalise the size of the design space. As a result, aerodynamic optimisation w.r.t. positions of all control points ($n = 1944$ design parameters) achieved significant cost function improvement.

6.4 Summary

This chapter addressed the challenges for the CAD-based shape optimisation of industrial components and showcased them for the TUM DrivAer vehicle test case. The baseline CAD model consists of multiple NURBS patches not suitable for the modification; the parametric model of the vehicle does not exist. One finds that the Shape Optimisation Framework is a useful tool for quick CAD model re-parametrisation. The differentiated OCCT was used to create a new suitable CAD topology and design space, while the gradient-based method addressed its high-fidelity fitting to the original CAD surfaces (mesh-to-CAD/CAD-to-CAD geometric optimisation). While here the implicit parametrisation was considered (NURBS surfaces), a similar approach can be used for the re-parametrisation of parametric CAD models. This automatic fitting procedure is an essential development for industrial CAD modelling applications, where often manual model calibration is considered in the reverse engineering and mesh-to-CAD conversion problems.
The re-parametrised surfaces of the TUM DrivAer mirror were further modified by the framework to minimise its aerodynamic noise. A new CFD solver was incorporated into the framework, where the cost function for the low-frequency noise was formulated with a differentiated surrogate aeroacoustic model. The suitable CAD design space was shown to be a critical factor that enabled considerable cost function improvement.

Chapter 7

Aerodynamic Optimisation of CRM Wing-Body Intersection

CAD models are usually created from several independent parts which are then combined using CAD Boolean operations (fuse, common, cut, etc.). In most cases, these operations perform surface-surface intersections to construct the final shape. As a result, the final CAD model includes a collection of trimmed patches. The current NURBS-based optimisation approaches are applied directly to the existing CAD topologies, they do not modify the locations of the trims (intersection curves) and therefore limit the design space. To alleviate this, a technique for the recalculation of the patches intersections is included into the Shape Optimisation Framework. This is an important step towards greater integration of CAD into design optimisation loop: a complementary mesh morphing technique that accommodates occurring changes in CAD topology (intersections/Boolean operations) is devised. The algorithms are built on top of the differentiated OCCT and hence provide necessary CAD sensitivities.

The methodology is applied to redesign the wing root fairing of the NASA Common Research Model (CRM) aircraft and minimise its drag. The baseline fairing-wing topology is discarded and the new optimal intersection between the fairing and the wing is determined. A similar parametrisation technique was successfully used in

the author's work for the DLR F6 aircraft geometry [129]. Here, a more complex AD-enabled OCCT CAD system is incorporated, which provides a general method valid in a case of non-trivial intersection curves [90].

The chapter is organised as follows: first section introduces the CRM test case, Section 7.2 describes a geometric pre-processing step, namely fairing re-parametrisation, that ensures the valid wing-fairing intersection exists. Section 7.3 outlines the algorithm for the recalculation of the intersections and the mesh morphing technique necessary during the optimisation. The differentiation of the process is described in Section 7.4, followed by the application of the method for the aerodynamic optimisation in Section 7.5.

7.1 NASA Common Research Model test case

The NASA Common Research Model resembles the shape of a typical transonic aircraft, as seen in Fig. 7.1. The test case was proposed by AIAA[1] in several Drag Prediction Workshops [11], which concentrated mainly on the assessment of the existing computational tools and methods for aircraft aerodynamics.

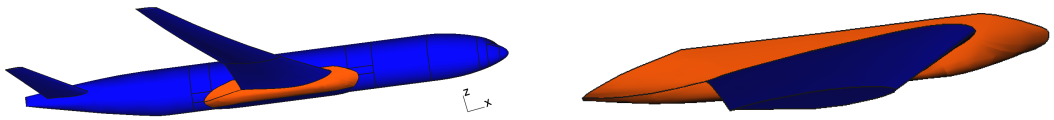

Figure 7.1: Left: Geometry of the NASA CRM wing-body model (NURBS surfaces); right: fairing subject to design changes (in orange).

Here, the application of the CAD-based optimisation technique is considered for the CRM drag minimisation (freestream Mach number $Ma = 0.85$ and 2.5 degrees angle of attack). Since the focus of this work is mainly on the novel CAD technique, the aerodynamic constraints, e.g. constant lift, are not considered in the optimisation.

[1] American Institute of Aeronautics and Astronautics

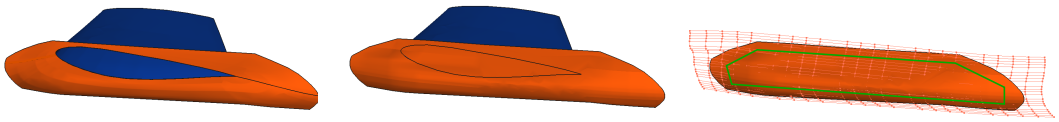

Figure 7.2: Left: Original trimmed fairing from the inside; middle: untrimmed re-approximated fairing; right: 22 × 22 fairing Control Point net and design control points (inside green area).

Geometry

The geometry of the NASA CRM aircraft is available in STEP format [11] and consists of a collection of NURBS patches (Fig. 7.1). Here, the NURBS-based redesign of the wing root fairing (the orange patch) is considered. All other patches (in blue), i.e. the complete fuselage and the entire wing, remain fixed and are not changed during the design process. Only geometric continuity constraints are imposed between the patches. The fairing surface is trimmed (has a hole, Fig. 7.2) at the junction with the wing, which poses several challenges for the fairing's design space definition.

7.2 CRM fairing re-parametrisation

The simpler approach, not followed here, is to apply the NURBS-based optimisation technique directly to the provided trimmed fairing patch. In this case, the G0 continuity constraint would be required on the common wing-fairing junction edges. This ensures that the geometry remains watertight during optimisation but consequently restricts the fairing's design space. In this scenario, only small areas on the fairing, that are not affected by the 'frozen' junction, could be deformed. Similar to the test cases in the previous chapters, the original baseline surfaces have a very dense distribution of control points (several thousand). Since the NASA CRM existed initially as a mesh-based test case [120], the over-parametrisation of the surfaces could be a product of a mesh-to-CAD conversion. As mentioned in

Section 6.2, these parametrisations are not practical for NURBS-based optimisation as they could capture oscillatory modes introduced by generally non-smooth CFD gradient fields. This may result in the undesirable 'wavy' surfaces.

To circumvent the disadvantages of the fairing parametrisation, the existing fairing face is discarded and a new surface is reverse engineered. The wing-fairing trim is removed with the surface filling functionality (`BRepOffsetAPI_MakeFilling` [8]) available in OCCT. The resulting untrimmed surface is a re-approximation of the original fairing patch and does not contain the hole. The surface accommodates a much smaller amount of control points ($22 \times 22 = 484$) on the fifth degree NURBS surface. As in Section 6.2, the gradient-based shape fitting could be applied for further high-fidelity re-approximation of the surface. In the remainder of the chapter, the re-parametrised untrimmed fairing patch is used.

The intersection of the re-parametrised fairing and the wing will be therefore determined during the optimisation process. As shown in Fig. 7.3 (left), the movement of the control points of the fairing towards the wing changes the intersection line between them. To ensure the fairing's deformation always returns a valid intersection, the wing was also extended into the aircraft's interior.

The continuity between the fairing and the rest of the fuselage is maintained by restricting the movement of the control points along the periphery of the patch. This ensures that the fairing and the fuselage remain G0 continuous. This manual step leaves 169 control points P free to move in any direction, which results in 507 design parameters (interior of the green region in Fig. 7.2). Restriction of additional rows and columns of control points can also impose G1 or G2 continuity (based on the degree of NURBS). Alternatively, one could employ the NSPCC/test-points approach (Sec. 4.3) to impose continuity constraints on the fuselage-fairing edges while using all fairing's control points.

In summary, the parametrisation allows modification of the single fairing patch with the control points P, while all other patches that belong to the fuselage and the wing remain fixed.

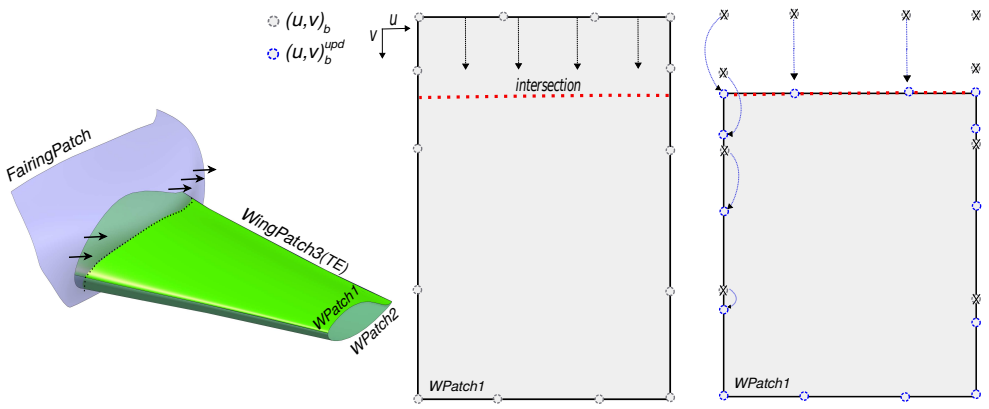

Figure 7.3: Fairing movement (left); initial position of the boundary mesh points in 2-D parametric space (middle) and the updated positions used for IDW (right).

7.3 Dealing with moving intersection

Variation of the design parameters modifies the fairing surface and hence the intersection with the fixed wing (Fig. 7.3). In particular, three intersection lines (each for three wing surfaces) modify the wing-fairing CAD topology. In the context of numerical simulation, this also requires a technique that can propagate and synchronise these geometrical changes with the computational grid. Several authors addressed this by deriving an explicit relation between the mesh points that belong to the intersection line and the positions of control points on a moving surface [82, 83, 129]. In these works, the updated intersection exists in terms of displaced 'corner/intersection' mesh points rather than as the geometrical entity. In this thesis, a general CAD-centric approach is developed that benefits from the functionality provided by the differentiated OCCT. The CAD system recalculates the intersection lines and then their geometrical description is used for grid deformation.

Algorithm for moving intersection

For the CRM case, the corresponding STEP file is parsed and the BRep data are extracted for the four design surfaces. Although only the fairing surface is allowed

to deform, we do consider the three wing patches (Fig. 7.2) as design surfaces, since the mesh on these surfaces will be modified due to the moving intersection lines. Firstly, point inversions algorithms are utilised to determine positions of the surface mesh points X_S on the design CAD faces. As a result, each Cartesian (3-D) mesh point acquires a pair of 2-D parametric coordinates *(u,v)* on the corresponding patch.

Secondly, OCCT is used to determine the topological bounds of each CAD face: a combination of edges and intersection lines that encircle the face (Fig. 7.3). On these boundaries, artificial mesh points (u_b, v_b) are generated and later used to control the displacement of the surface mesh points within each patch. Finally, any movement of the fairing triggers the following sequence of operations:

(i) OCCT recalculates all intersections and uses them to reconstruct the topological boundaries of all four design faces.

(ii) On each topological boundary (whether it is the intersection or the edge) the artificial boundary mesh points are redistributed and updated $({u_b}^{upd}, {v_b}^{upd})$. The constant arc-length ratio between each boundary point is maintained based on the 1-D parametrisation of the corresponding boundary curve. This helps maintaining uniform distribution of boundary points in 2-D parametric space (Fig. 7.3 (right)).

(iii) The inverse distance weighting mesh smoothing [125](IDW) is applied in 2-D space of each face to propagate boundary movement to the surface mesh points (u, v). Points (u_b, v_b) and $({u_b}^{upd}, {v_b}^{upd})$ are used as the boundary conditions for the method. With this approach, all inner points remain within the bounds of the updated face.

(iv) Finally, the updated parametric coordinates are used to calculate 3-D Cartesian surface mesh displacements and propagate them into the volume mesh also using IDW, as described in Subsection 2.2.2.

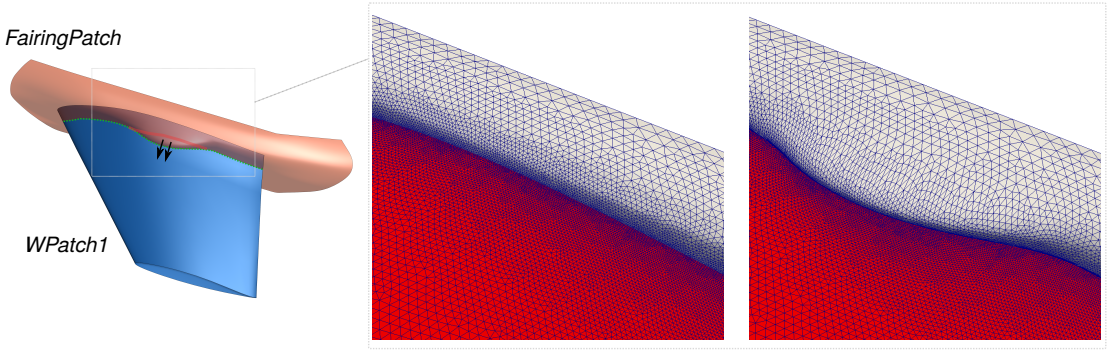

Figure 7.4: Fairing perturbation (left); original and perturbed surface meshes (right).

The algorithm described above has a hierarchical structure: one starts with mesh displacement in 1-D space of each topological boundary. Afterwards, these displacements are propagated in 2-D parametric space of each design surface. Finally, an updated 3-D Cartesian mesh is available. The method maintains constant mesh topology on each patch. For large design changes, this could deteriorate mesh quality and the re-meshing might be necessary. Alternatively, the development of mesh movement methods directly in the 3-D space can be beneficial. For multi-patch intersection cases, where mesh movement is no longer restricted within each patch, this would allow the surface mesh to 'slide' between adjacent patches, avoiding undesirable mesh stretches appearing for the large deformations.

In Figure 7.4 (left), several control points were deliberately perturbed in the vicinity of the wing junction, such that the fairing created a 'bump' on the wing. The results of steps (ii)-(iii) (2-D mesh smoothing) are shown for the parametric spaces of the fairing and one of the wing faces in Figures 7.5 and 7.6, respectively. As a consequence of this 2-D mesh movement, the corresponding 3-D surface mesh follows the changes in CAD geometry and accounts for the movement of the intersection line (Fig. 7.4, right).

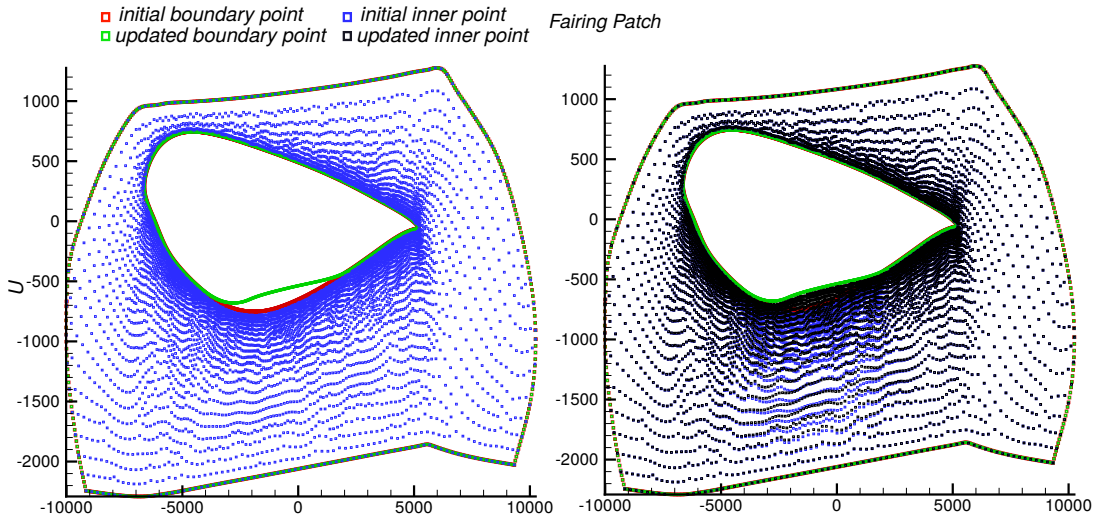

Figure 7.5: The original (left) and perturbed (right) surface meshes in 2-D parametric space of the fairing.

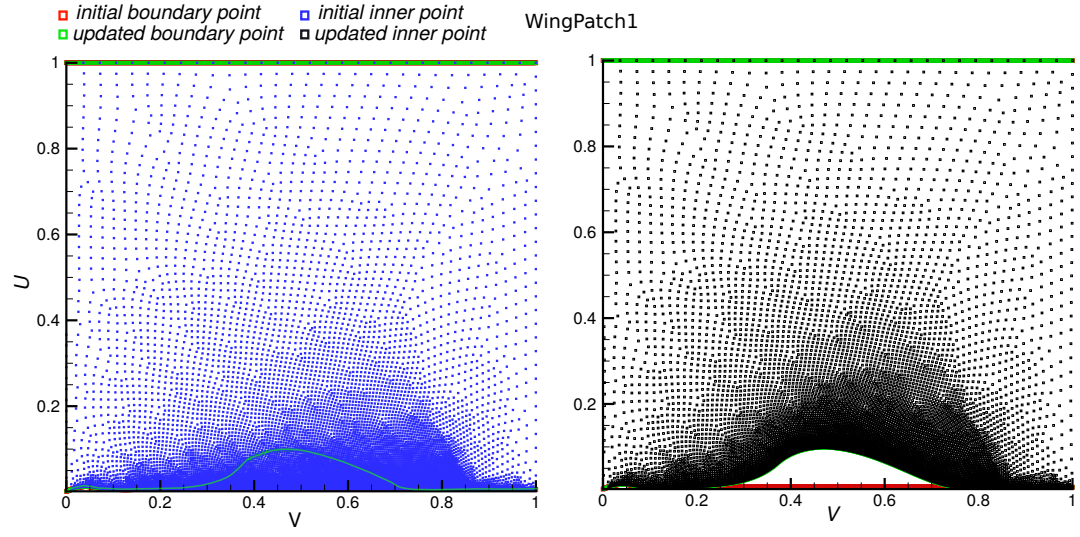

Figure 7.6: The original (left) and perturbed (right) surface meshes in 2-D parametric space of the bottom wing's surface (WPatch1).

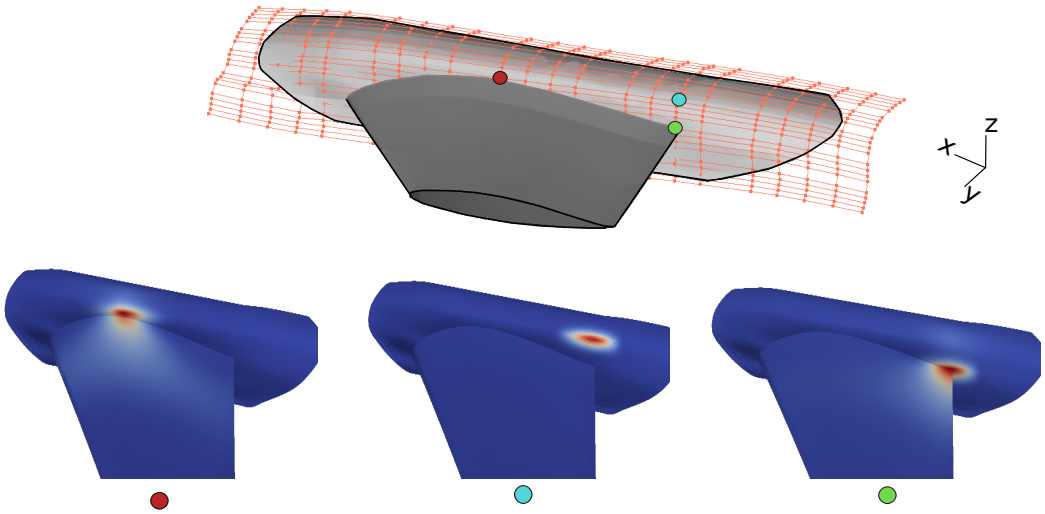

Figure 7.7: CAD sensitivities w.r.t. the positions of three different control points (y-component).

7.4 CAD sensitivities including intersections

The previous section described the algorithm for the synchronised CAD-mesh movement occurring due to finite deformation of the fairing surface. Since the intersection recalculation and the corresponding mesh morphing algorithm is built on top of the differentiated OCCT, the complete process (i)-(iv) is also automatically differentiated. Therefore, the code is capable of providing exact CAD sensitivities w.r.t. the movement of the fairing's control points on four design surfaces. In Figure 7.7, the CAD sensitivities are shown for three different control points. In all cases, the CAD sensitivities for the movement of the control points along the wing ($\frac{dX_{Sy}}{dP_y}$, y-direction) are considered. The variation of the first control point (red) is related to the finite step deformation discussed in the previous section. Due to the direct impact of the control point on the intersection line, the high sensitivity values are visible on both the fairing and the wing patch. As a result, the CAD sensitivity resembles the finite step mesh changes ('bump') on these surfaces (Fig. 7.4). The second control point (blue) is located further from the junction and, due to the locality of NURBS, does not influence the surface at the wing junction.

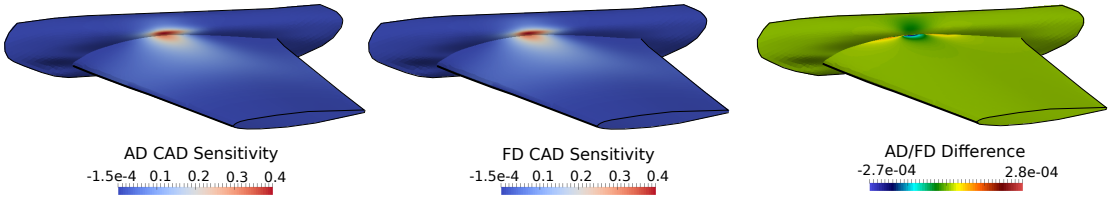

Figure 7.8: Comparison of CAD sensitivities computed by AD and FD w.r.t. fairing control point movement (y-direction).

Naturally, the non-zero sensitivities are located on the fairing surface only. The same sensitivity would be obtained for the non-intersecting fairing. Finally, the last control point (green) influences the fairing intersection with all three wing surfaces (top, bottom and the trailing edge). Consistently, the non-zero CAD derivatives are present on all design surfaces.

The CAD sensitivities were also compared with the finite differences (central scheme) results and showed mutual agreement, thus providing confidence in the differentiation of OCCT and the algorithms presented in this chapter. AD/FD comparison was conducted for several design parameters and is shown in Fig. 7.8 for the previously defined 'red' control point. Similarly to the TUB Stator blade, the acceptable maximum discrepancy between the AD and FD sensitivities is achieved ($O(10^{-4})$) but can be degraded by a poor choice of FD step size. Since the CAD intersection algorithms are computationally expensive, finding derivatives w.r.t. all 507 design parameters with FD is time-consuming. On the contrary, the block-vector AD mode allows finding all derivatives in a couple of minutes and does not require any finite step size calibration. One has to note, that the developed block-vector AD mode was the most efficient solution in this case. OCCT in the reverse AD mode or forward vector mode required a large amount of memory to build traces or vectorised computational graphs for the complex intersection/mesh morphing algorithm. Further OCCT structure exploitation has to be performed to enable the use of these modes for the test case.

7.5 CRM optimisation

The algorithm introduced in the previous sections provides CAD sensitivities necessary for the gradient-based optimisation of the CRM's fairing. The corresponding parametrisation accounts for the changes in the wing-fairing topology and the mesh deformation. In this section, the Shape Optimisation Framework is used to reduce the drag of the NASA CRM aircraft model. Since the primary focus of this chapter is the demonstration of the complex CAD-based method with fully differentiated design chain, simplified flow and optimisation goal are considered. To accelerate aerodynamic simulations, the compressible Euler equations are solved using QMUL's in-house CFD solver mgopt, therefore the viscous effects are not considered. The objective function is the drag on the design surfaces (fairing and the adjacent wing patches) and no aerodynamic constraints are imposed.

The CAD geometry of the complete CRM model with the re-approximated fairing was used to produce the computational mesh. The wing-body configuration without nacelle and pylon was used [120]. The CFD mesh produced using the provided CAD geometry has 1.2 million tetrahedral elements. Calculations are performed at the defined Mach number $Ma = 0.85$ and 2.5° angle of attack.

At each design step, the steady-state flow and the adjoint solutions are first computed, the CFD sensitivity is assembled with the aforementioned CAD derivatives on the design surfaces. Finally, the gradient of the objective function is provided to the steepest descent optimiser which returns the updated positions of the fairing's control points.

Usually, the goal of the fairing optimisation, as shown in the authors' work on DLR F6 case [129], is to suppress the flow separation at the wing-fairing junction near the trailing edge. Since the viscous effects are not considered in the CFD model, this phenomena is not present for the CRM. In the inviscid case, the drag is mainly caused by the pressure/wave drag (presence of a shock at the transonic flight condition) and by the lift-induced drag.

Obtained CRM's pressure coefficient contours c_p for the baseline geometry are

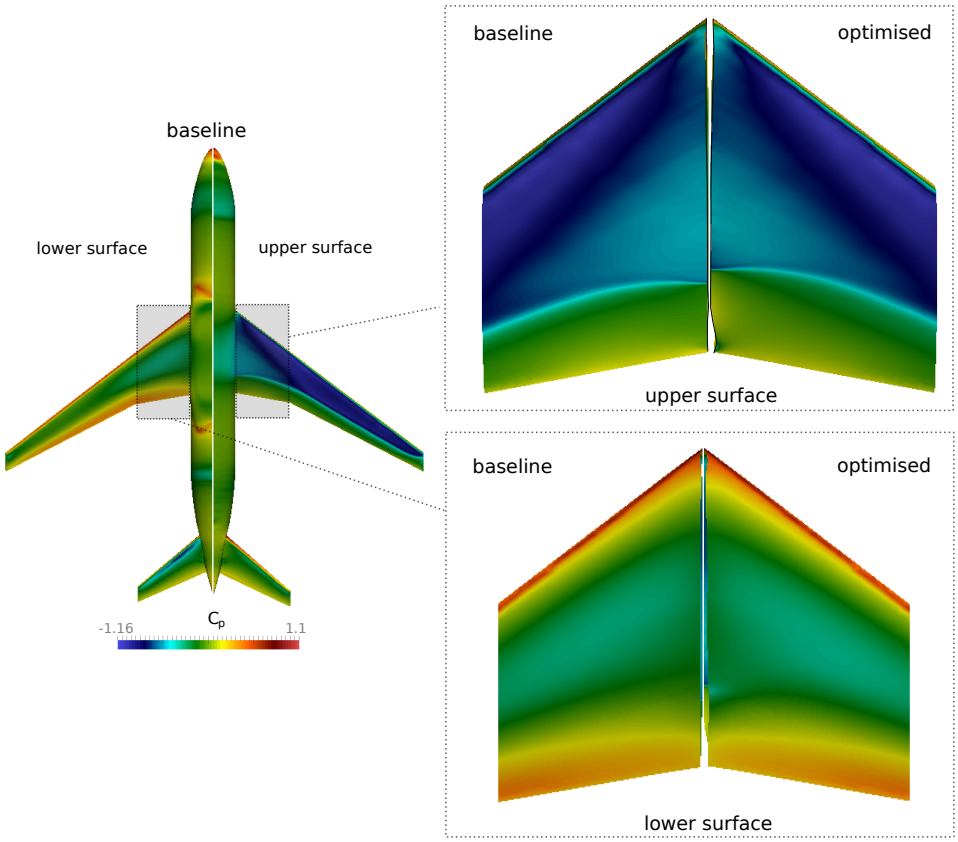

Figure 7.9: Pressure coefficient c_p contours at the upper and lower surface (left); optimised shock position in the vicinity of the wing-fairing junction (right).

shown in Fig. 7.9. After eight optimisation iterations, the optimiser achieves 7% drag reduction, resulting also in a smaller lift (Fig. 7.11). The total drag reduction for the optimised geometry can be attributed to the movement of the shock position upstream as well as a larger low pressure area on the lower surface of the wing (expansion of green colour contour region in Fig. 7.9, right). This leads to the decrease in the lift and lift-induced drag, also visible from the Mach number and the pressure coefficient redistribution along the wing's chord (Fig. 7.10).

In Figure 7.12 the difference between the original and the optimised fairing is shown. The recalculation of the new wing-fairing topology at each optimisation step enabled a rich design space, not restricted by the 'frozen' intersection line. Large deformations are visible along the wing (y-direction) with non-zero displace-

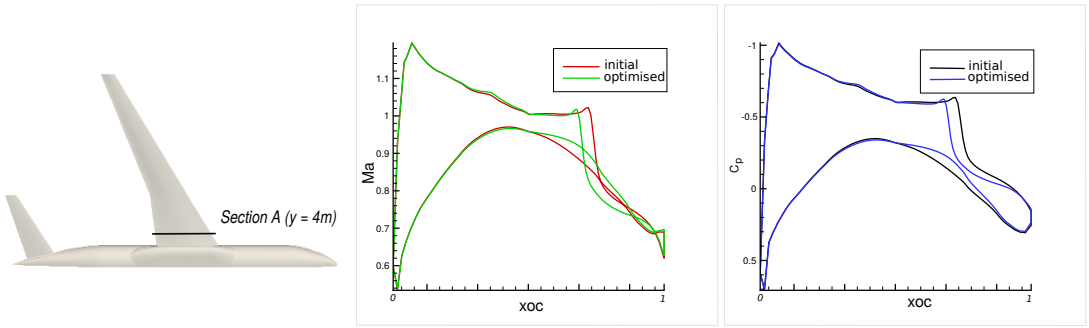

Figure 7.10: Initial/Optimised Mach number and c_p distributions at Section A.

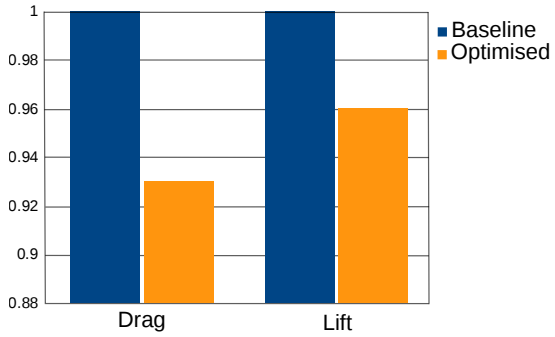

Figure 7.11: Drag reduction for the optimised fairing.

ments at the intersection line, which allowed the flow modification (different pressure distribution) and reduced the cost function. While these results were obtained for the simplified flow conditions, they demonstrate the advantages of including the intersection recalculation into the optimisation loop.

Although the inviscid flow calculations can be useful for preliminary aircraft design [62], higher fidelity CFD models can yield more accurate but also modified flow field (changes to the position of the shock, flow separation) and hence lead to a different optimisation result. As long as these models provide CFD sensitivities, they can be coupled with the framework and incorporated into the design loop. Without limitations, the same CAD-based technique can be applied with viscous, turbulent, steady and unsteady flow conditions. Also, the Shape Optimisation Framework can be utilised to perform drag minimisation with constrained aircraft's lift along with other aerodynamic or geometric constraints. This can ensure the efficient improvement of the aircraft's lift-to-drag ratio. Additionally, the

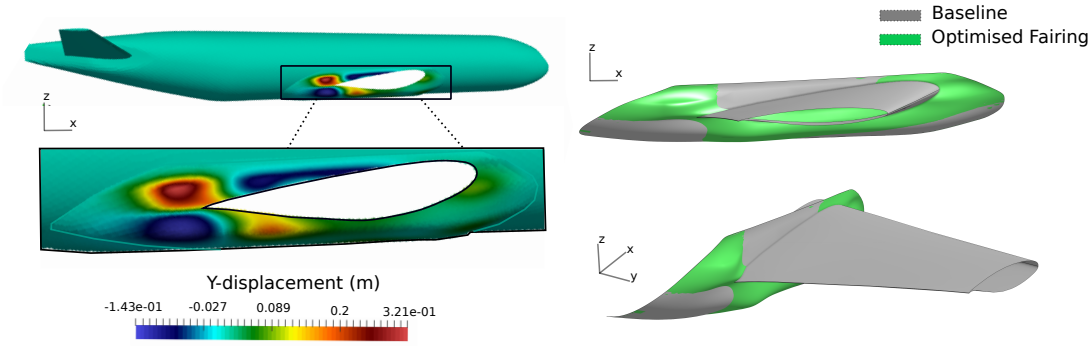

Figure 7.12: Fairing deformation mapped on the baseline geometry (left); initial and updated design surfaces (right).

demonstrated parametrisation technique can be applied to other design problems with complex geometries, such as optimisation of aircraft's wing-pylon intersection (optimal pylon's mounting position) [82].

7.6 Summary

A novel CAD-based algorithm that enables the recalculation of patch intersections during gradient-based optimisation was presented in this chapter. At every design iteration, OCCT perturbs CAD model surfaces, recalculates cross-patch intersection lines caused by this movement and uses their geometric description to deform the computational grid. The differentiation of a complete OCCT CAD kernel was essential for this task, as the derivatives of the surface-surface intersection algorithms are efficiently calculated with the block-vector mode of AD.

The inviscid flow conditions were used to drive the drag optimisation of the NASA CRM aircraft and showcased the potential of the CAD-based method. The new fairing and the intersection line with significant deformation along the wing profile were found to reduce the cost function.

The integration of the CAD system intersection capabilities into the optimisation loop enables the application of the method to the CAD models with multiple non-trivial intersecting patches and parametric CAD models generated with Boolean

operations. This is an important step towards CAD-based optimisation of increasingly complex industrial components.

Chapter 8

Conclusions and Future Work

A major contribution of this thesis is in bridging the gap between CAD and simulation tools in industrial shape optimisation workflows. The developed Optimisation Framework for CAD models brings together their geometrical description, corresponding computational grids and provides gradients of aerodynamic or geometric cost functions w.r.t. model's design parameters. This allows maintaining CAD models as input-output data in the gradient-based optimisation loops, consistently offering a tighter integration and automation of industrial CAx components. The framework's 'master-CAD' paradigm simplifies the coupling of different simulation tools and optimisers, streamlines aerodynamic optimisation process and creates possibilities for multi-disciplinary (multi-department) collaboration.

Notably, the developments in this thesis rely heavily on the capabilities of Automatic Differentiation - for the first time, the adjoint CFD simulation tools and the developed CAD software composed a fully differentiated design chain. The differentiated OpenCascade Technology (OCCT) CAD kernel is the cornerstone ingredient of the chain - it provides the necessary CAD sensitivities efficiently, accurately and robustly. This is an enabling factor for shape optimisation in large design spaces. Two different techniques for parametrisations were developed: conventional parametric CAD models or automatic NURBS-based parametrisations can be used depending on the characteristics of the test case.

The efficiency and flexibility of the underlying CAD design spaces is demonstrated by the successful aerodynamic optimisation of three different industrial-scale test cases.

8.1 Differentiated CAD kernel

The OCCT differentiation was achieved with the injection of the ADOL-C library into the original source code. The kernel's object-oriented architecture was well-suited for the AD by operator overloading and both forward and reverse mode versions of OCCT are available. The developed block-vector AD mode presents an efficient way of computing CAD derivatives and allows users to achieve the compromise between code's computational efficiency and memory consumption. The approach was successfully applied to parametrisations with several hundreds of design parameters (Chapters 5-7) which enabled large-dimensional design space explorations. In these cases, the differentiated OCCT alleviates the bottlenecks of the commonly used finite difference differentiation approach and outperforms it in both accuracy and efficiency. CAD sensitivities of NURBS-based parametrisations and complex geometric algorithms used in CAD lofting and surface-surface intersections were validated.

The differentiation of OCCT also shows the power of AD: the technique accounts for the sophisticated algorithms with deep computational graphs and does not necessarily require a profound understanding of the underlying code structure. This makes the differentiated OCCT easily extensible - derivatives of many CAD-based algorithms, including those not discussed in the thesis, can be obtained. Codes that use OCCT's data structures also benefit from automatic derivative information.

8.2 Parametrisations for aerodynamic shape optimisation

Two parametrisation techniques - implicit NURBS-based and explicit parametric-based - were developed for aerodynamic optimisation of industrial components. The choice of either of the parametrisation for optimisation of an arbitrary CAD model is case-dependent since both approaches perform design explorations in different design spaces. The parametric CAD models are useful for the applications when decent parametrisation practices are established through the previous engineering experience. The NURBS-based approach could then serve as the complementary or the alternative technique, which is also advantageous for the optimisation of non-conventional components. In the proposed hybrid approach, both parametrisation techniques were used simultaneously to optimise different parts of a single CAD component.

For both techniques, the recipes for imposing manufacturing constraints were developed. Access to the OCCT allows users to define and measure the constraints directly on the CAD model geometry. The available optimisers can use the automatic gradients of the corresponding constraint functions. The storage of the constraints in the standard CAD files and hence the possibility for their visualisation and inspection was demonstrated for the NURBS-based approach.

Additionally, a novel application of NURBS-based technique was proposed for CAD topologies that depend on the intersections between different CAD parts (e.g. the result of Boolean operations). Recalculation of the intersections and the complementary mesh morphing algorithms, implemented in OCCT, allowed gradient-based optimisation with non-constant topologies. This presents an important step towards automated optimisation of complex CAD parametrisations.

8.3 CAD shape fitting

In this work, the Shape Optimisation Framework has proven to be an effective tool for CAD model re-parametrisation and fitting. The discrepancy between a given shape and OCCT parametrisation can be expressed by distance metrics, such as least squares differences between pairs of surface points on both models. The corresponding gradient-based distance minimisation offers a CAD-to-CAD or CAD-to-mesh morphing procedure and enables high-fidelity algorithmic fitting. The latter presents an effective CAD-based technique for the commonly occurring reverse engineering tasks, such as transfer of scanned 3-D data/point cloud to CAD, where the process often relies on manual CAD model calibration. Differentiation of complete CAD system makes it possible to consider in the fitting task geometric constraints, build useful distance metrics that besides point-to-point distance include other geometric properties (point's normal, curvature, etc.).

8.4 Future directions

The thesis proposed successful integration of differentiated CAD system into gradient-based optimisation chain for aerodynamic design. This also opened several potential areas for future research, both in further OCCT development and in utilisation of CAD gradients in other applications.

- OCCT structure exploitation: the proposed type substitution approach (Section 3.2) for the integration of ADOL-C library into OCCT was chosen as the fastest and most convenient method for differentiation. However, in many cases unnecessary resources are allocated for the library's `adouble` variables that do not influence derivatives. To address that, further analysis of algorithms and changes to their structure can be performed to improve the computational efficiency of the differentiated OCCT.

- Further applications with complex CAD models: although in the aerody-

namic optimisation one often focuses on the particular part (e.g. single turbomachinery blade), the differentiation of the CAD software allows treating more complex parametrisations with multiple parts (e.g. subject to Boolean operations) as a complete system. With the proposed developments for the intersections recalculation (Chapter 7) and the maturity of simulation tools, the optimisation of increasingly complex components can be performed.

- Aerodynamic and multidisciplinary optimisation: higher fidelity CFD models can be incorporated into the framework for accurate performance prediction. Since many industrial flows are unsteady (turbomachinery, wind turbines), the corresponding adjoint formulations can be used for time-accurate optimal design. In addition to aerodynamic shape optimisation, the coupling of the differentiated OCCT with acoustic, structural, or heat transfer solvers, as well as the use of the kernel for the uncertainty quantification problems, can be investigated. The dependency of the simulation tools on the common 'master-CAD' geometry can be used to drive the multidisciplinary optimisation process robustly.

- Geometric optimisation: besides aerodynamic applications, CAD gradients were useful in the minimisation of purely geometric cost functions (distance metrics). A similar approach can be applied for collision detection, e.g. minimisation of the intersection area between two CAD parts (Appendix C).

  Several studies suggest the possibilities for quantification of aesthetics and attractiveness of certain geometrical shapes [79, 118]. Optimisation of the shape's curvature or, if available, more sophisticated 'style' functions could be an interesting avenue for design with CAD derivatives.

- Data-driven applications: although the differentiated OCCT does not require a high computational cost to obtain derivatives (enabling work with rich parametric models or refined NURBS surfaces), in many cases controlling the shapes with a handful of parameters could be beneficial. The CAD

gradients of these parametrisations can be used in the CAD space dimensionality reduction, as proposed in the Active Subspaces method [96]. The latter can be also considered as a CAD parametrisation coarsening technique (opposite to NURBS knot refinement), where just a few combinations of the original parameters can represent most of the shape's variability.

Additionally, the differentiated OCCT can be used in gradient-enhanced surrogate modelling for CFD simulations or other applications that build data-driven models with CAD input.

- Software with gradients: the thesis has proven the feasibility of AD integration into an existing large-scale software system. Since gradients of complex functions augment the code with additional information valuable in many scenarios, the OCCT differentiation techniques can be useful in obtaining derivatives of other C++ codes.

Appendix A

CAD Workflows and OpenCascade Technology

In this appendix, we overview typical CAD workflows and corresponding geometrical principles behind them. In the context of shape optimisation, these workflows define model parametrisation (design space) and encapsulate algorithms subject to differentiation. The appendix also describes the open-source CAD kernel OpenCascade Technology (OCCT) and its modelling capabilities.

A.1 CAD systems and workflow

In engineering industries a CAD system is a key tool for the design and development of a new component. The companies are using CAD software provided by different vendors such as Autodesk, Dassault Systems, OnShape, OCCT, Siemens PLM Software, to name a few, or utilise their specialised in-house CAD implementations. Most of these tools are based on common geometric principles and provide similar capabilities. Usually, the CAD tools come with a convenient GUI (front-end) that can simplify and accelerate the development and revision of the component. In the back-end, there is a kernel which performs geometric computations, stores corresponding data-structures, etc. The open-source CAD modeller FreeCAD [3]

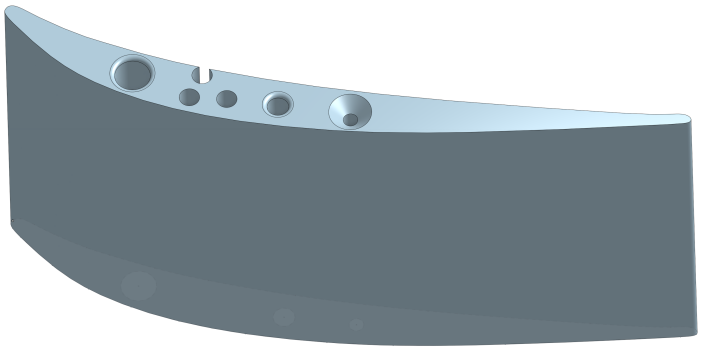

Figure A.1: Blade component produced in OnShape.

is a good example of such architecture, where the GUI is built on top of the OpenCascade Technology. OnShape proposes a cloud-based GUI solution, with back-end running on the remote server [5].

To review typical CAD workflows and algorithms, the OnShape GUI is used to construct a parametric blade (Fig. A.1), which resembles the shape of TUB TurboLab Stator blade (Sec. 4.1). Since the geometry of the TUB blade is provided in the STEP format, here the parametric CAD model is reverse engineered.

A.1.1 Parametric modelling

Sketches and 2-D profiles

A common approach to start the development of a new CAD model is to define a sketch on a plane - a 2-D profile of the component. CAD tools allow creating several types of sketches by using primitive profiles e.g. circles, rectangles, etc., or more generic B-spline curves [102]. These profiles can be used then as a base for the final 3-D model construction. In the context of parametric modelling, the parameters of these sketches (circle radius, rectangle location, position of the B-splines control points) determine the appearance of the sketches and the final shape. Certain constraints such as specified sketch dimension, parallel/perpendicular lines, etc., can be imposed. In Figure A.2, four independent sketches were constructed using B-spline curves and circular profiles.

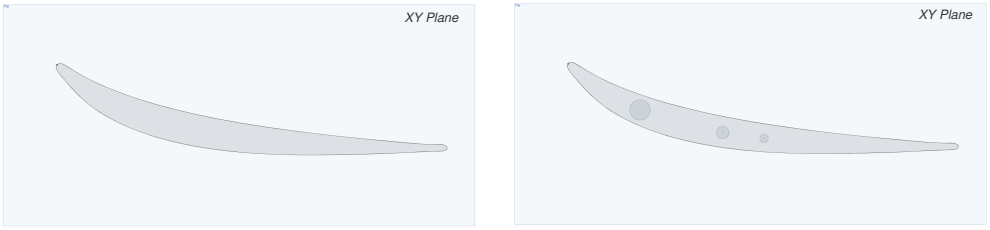

Figure A.2: B-spline blade profile and three circular sketches with different radii on XY plane.

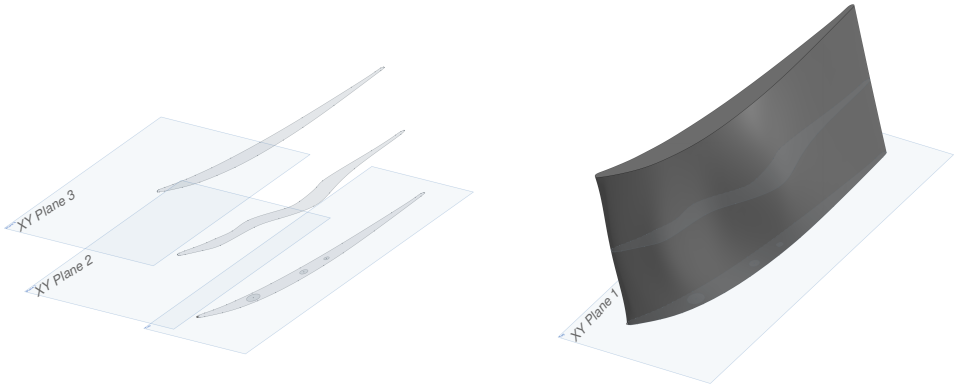

Figure A.3: Cross-sectional design approach: lofting through three blade sections.

From 2-D sketches to 3-D

Lofting is a CAD operation which takes as an input several profiles/slices and spans a 3-D shape through them. Geometrically, the lofting operation is similar to surface interpolation, where slices correspond to the interpolation data nodes. In Figure A.3, the lofted model is built through the initial and distorted blade profiles located on three offset XY planes. This method constructs rich design space, useful for shape optimisation: each section parameter could be independently altered causing local 3-D shape changes in the vicinity of the section parameter. Later, we also refer to this technique as a cross-sectional design approach.

Another way to create a 3-D model from the pre-defined sketch is to use a Sweeping

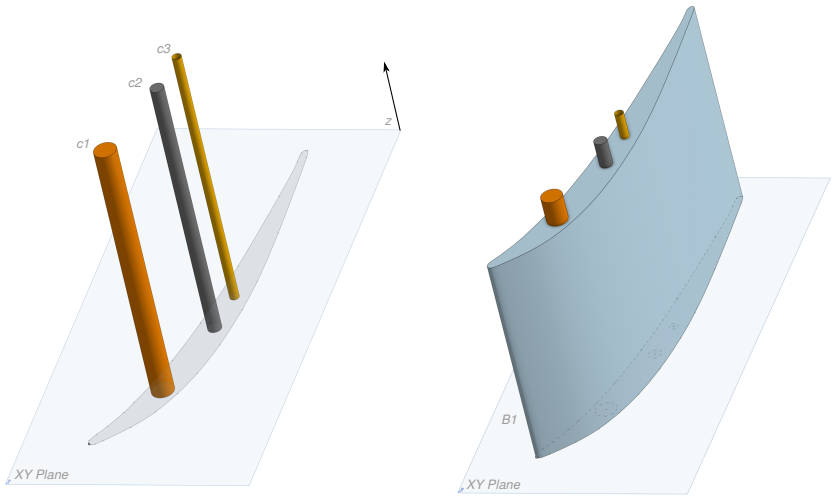

Figure A.4: Circular and blade sketches extruded along *z-axis*.

operation. We can define a path-line (in most cases a guiding B-spline curve) which evolves the slice in the 3-D space. Similarly, an Extrusion operation is the sweeping along the line (usually coordinate axes). In Figure A.4, both circular and blade sketches are extruded along the z-axis. In comparison to lofting, sweeping is generated with a slightly different and more restricted design space. The 2-D slice remains fixed in 3-D and could only be moved 'rigidly' along the guiding curve. Therefore, the design space consists of the baseline slice parameters and control variables of the guiding curve.

In some cases, both lofting and sweeping operations may fail to generate the desired shape. Certain prerequisites on the design space usually resolve the problem: similarity of the slices for lofting, or milder curvature of the sweeping path-line simplifies the underlying geometric algorithms and ensures successful 3-D shape construction.

Boolean operations

In the previous subsections, we considered ways to construct 3-D shapes using CAD: three cylinders (*c1, c2, c3*) with different radii and the 3-D blade (*B1*)

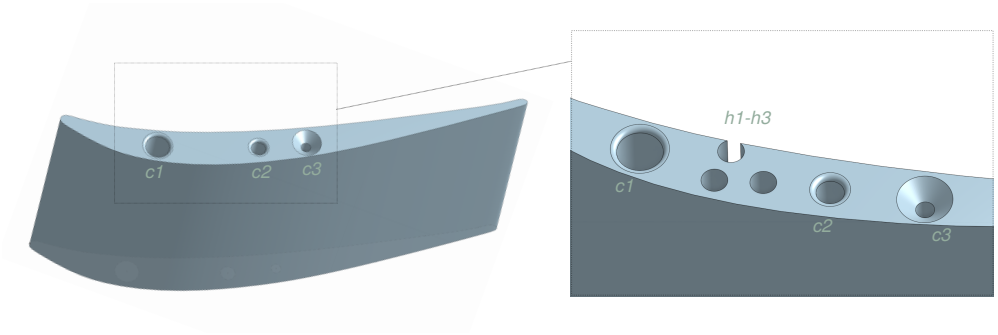

Figure A.5: Left: Result of Subtraction Boolean operation; right: fillets and chamfer on *c1, c2* and *c3* edges.

were created by extrusion of the sketches (Fig. A.4). CAD tools also allow direct creation of simple 3-D primitives such as spheres, cylinders, cones, etc., with the corresponding geometric parameters. At this stage, all these shapes exist as independent parts. CAD Boolean operations (Union, Subtraction, Common) allow to add, subtract or intersect these 3-D shapes and generate a more complex single model. Geometrically, the Boolean operations rely on the surface-surface intersection algorithms.

Figure A.5 shows the result of the subtraction operation between the blade and three cylinders (*c1-c3*). Similar 'holes' were generated by the subtraction operation between the blade and cylindrical primitives (*h1-h3*). Additionally, the fillet (rounded edges) and the chamfer operations were applied on the edges of cylinder holes *c1, c2* and *c3* respectively. In Figure A.6, the blade is merged with a cylindrical casing with the Union operation. Here, the CAD tool calculates the intersection edges to construct the final shape.

A.1.2 Absence of parametric standard: differentiation perspective

The operations outlined above correspond to the parametric modelling approach. The final CAD model depends on the features that define parametrisation: sketches, primitives, 3-D operations and their design parameters. CAD systems store the

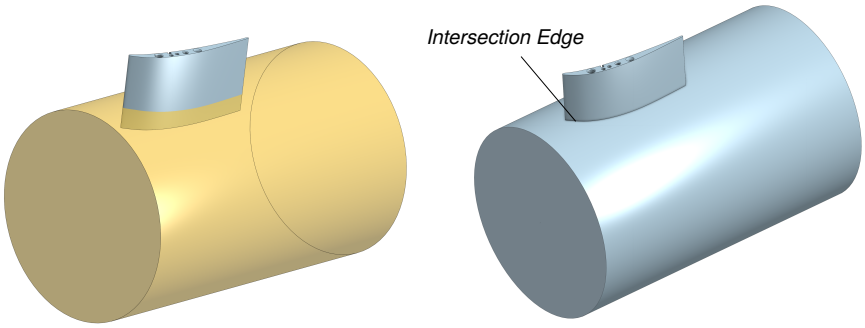

Figure A.6: Left: two independent components - blade and cylindrical casing; right: result of the Union operation between casing and blade.

sequence of these features in the so-called feature-tree. Therefore, the same tree could be used to regenerate the model with updated parameters or additional features. Unfortunately, there is no strict agreement on a generic standard for such feature-tree. Differences between CAD tools and their approaches to modelling make it complex to find the non-ambiguous parametric format. The absence of the standard causes challenges for the exchange of parametric models. To this end, some progress was achieved in the reconstruction of the primitive-based feature-trees (Constructive Solid Geometry) from generic geometric data such as point clouds or meshes [28, 44, 110]. The procedure usually includes segmentation (primitive candidates are proposed), fitting (primitive parameters are determined) and feature-tree generation. However, automatic reconstruction of complex parametric trees remains challenging and is often solved with manual reverse engineering.
Differentiated CAD system inherits this limitation: if the parametrisation procedure is not explicitly available - the system cannot provide derivatives of the parametric models created in another CAD tool. However, differentiated CAD can be used for the CAD-to-CAD fitting purposes (Sec. 6.2).

A.2 Topology and Geometry in CAD

Another way to look at a CAD model is not as a set of parameters and features that construct it but instead to consider the resulting shape. Within CAD system, the shape can be defined with the Boundary Representation (BRep) format and the concepts of Topology and Geometry. The BRep could be stored in the standard CAD files (STEP, IGES) and therefore could be transferred between different CAD systems.

Topological entities (Faces, Edges, Vertexes) define the structure of the shape and its mutual connections. Topology data structure allows the user to iterate through all faces, find the edges that encircle a given face, the beginning (first vertex) and the end (last vertex) of the edges, etc.

All topological entities have their geometric counterparts (Surfaces, Curves, Points). In most cases, surfaces and curves are represented with Non-uniform Rational B-splines (NURBS) or canonical surfaces (Plane, Sphere, etc.). NURBS curves and surfaces are defined as

$$C_S(u) = \sum_{i=0}^{n} B_i(u) P_i \,, \tag{A.2.1}$$

and

$$X_S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_{i,j}(u,v) P_{i,j} \,. \tag{A.2.2}$$

Here, P_i and $P_{i,j}$ correspond to the control points of NURBS curve and surface, (u, v) are their coordinates in the bounded parametric space. Therefore, a pass through the coordinates of the parametric space returns corresponding Cartesian points $C_S(u)$ and $X_S(u, v)$ that belong to the curve and the surface, respectively. In the equations above, B_i and $B_{i,j}$ are defined with B-spline rational basis functions as

$$B_i(u) = \frac{N_{i,p}(u) w_i}{\sum_{k=0}^{n} \sum_{l=0}^{n} N_{k,p}(u) w_k} \,, \tag{A.2.3}$$

and

$$B_{i,j}(u,v) = \frac{N_{i,p}(u) N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n} \sum_{l=0}^{n} N_{k,p}(u) N_{l,q}(v) w_{k,l}} \,. \tag{A.2.4}$$

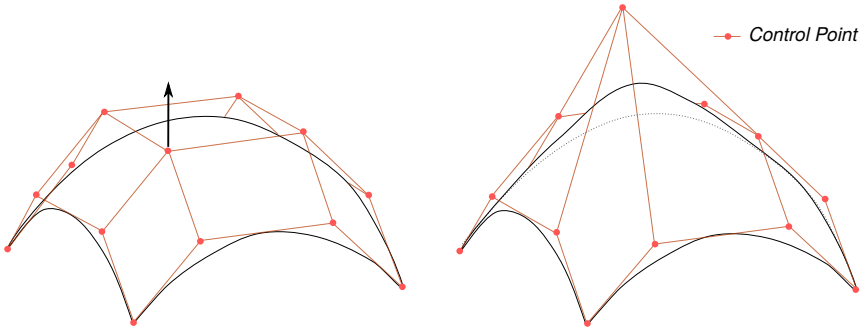

Figure A.7: NURBS surface and influence of Control Point movement.

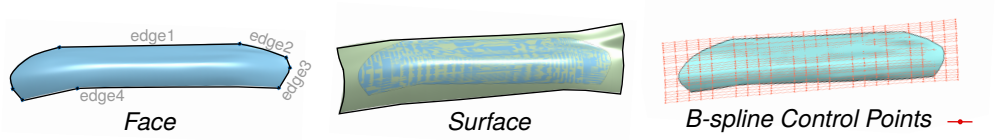

Figure A.8: Topological and Geometrical representation of the shape.

Here, w_i and $w_{i,j}$ are NURBS weights and $N_{i,p}(u)$ correspond to p^{th}-degree B-spline basis function. The positions of these control points and their weights locally influence the resulting geometry of the NURBS surfaces (Fig. A.7) and curves. For further details on NURBS the reader is referred to the book by L. Piegl [102]. With access to all topological and geometrical entities, one can obtain the position of every point of a given CAD model. Figure A.8 shows one of the faces of the NASA CRM test case (wing-fuselage fairing) which also demonstrates the difference between Topology and Geometry. The face is encircled by several edges (blue) and is considered as the valid CAD shape. The underlying geometrical NURBS surface exceeds these bounds and represents rectangular patch (green).

A.3 OpenCascade Technology CAD kernel

OpenCascade Technology (OCCT) is a powerful open-source geometric CAD kernel [6]. It includes most of the functionality available in modern CAD systems and could be considered as the competitor to the commercial tools available on the market. OCCT does not provide a built-in graphical user interface (GUI) but

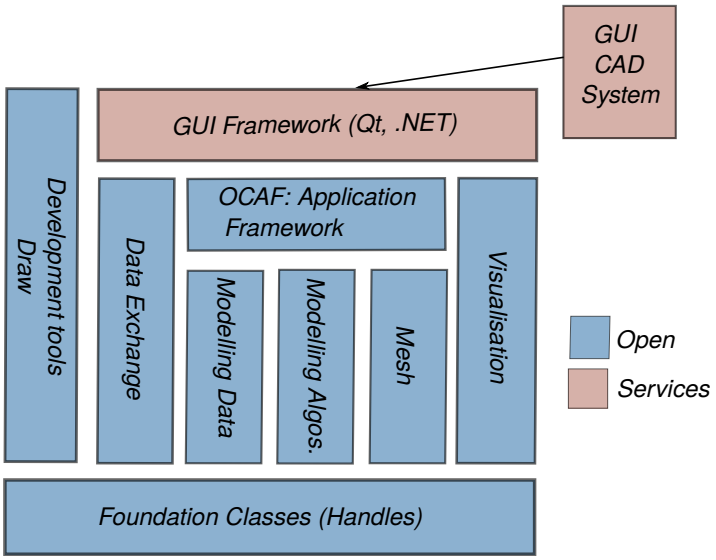

Figure A.9: OCCT architecture: open-source geometric kernel modules and services for CAD visualisation.

instead, it proposes a number of classes which could serve in the purpose of creating one (Fig. A.9). The OCCT distribution has a visualisation and testing tool 'Draw', which is useful for the quick inspection of the modelled geometric shapes. New versions of OCCT are still released quite regularly, but the core part of the software is not subject to dramatic changes, localising major updates in the parts responsible for the visualisation. OCCT is portable on Windows and Unix-based systems with an automated installation process. In addition, OCCT has detailed documentation and a number of helpful tutorials.

One of the biggest advantages of the software is its open-source nature. Access to the source code creates the possibilities for further independent development and modification. This is the key ingredient that makes OCCT suitable for automatic differentiation.

Most of the code is written in object-oriented fashion in C++, with some older subroutines also using C (Fig. A.9). Foundation Classes compose the base of the OCCT kernel and include essential geometric and math libraries, definitions of OCCT-specific standard data types and useful C++ extensions (e.g. `Standard_Real`,

`Standard_String` - OCCT type substitution for `double` and `string`, `Handle` - OCCT smart pointer implementation, `TCollection` - generic collections, etc.). Some of these classes could be considered as a legacy code since the underlying concepts are fully supported by the modern C++ (e.g. smart pointers, collections, etc.).

The Modelling Data module supports high-level CAD operations described in the previous subsections. Here, OCCT defines Topology and 2-D/3-D Geometry data structures: `TopoDSShape` - main topology class that can store any CAD model/shape, corresponding inherited topological classes `TopoDS_Face, TopoDS_Edge` etc., and their geometric complements `Geom_Surface, Geom_Curve`. Two other packages `TopExp` and `BRepTools` provide tools to iterate through the shape's topological entities and access their geometric information (e.g. retrieve 2-D parametric coordinate `gp_Pnt2d` of the point on the Face/Surface or its Cartesian coordinates `gp_Pnt3d`). In the Algorithms module, OCCT proposes a wide range of methods for parametric shape construction and underlying geometric algorithms: construction of primitives, boolean operations, interpolation, conversion of surfaces to B-splines, point inversion (point projection, closest point), surface-surface intersections, to name a few.

OCCT also includes subroutines which are responsible for efficient data exchange, shape healing and read/write operations for standard STEP, IGES files. Therefore, a shape modelled in another CAD tool could be exported to the OCCT `TopoDS_Shape` object and then manipulated by means of OCCT.

For the convenience of the reader, several OCCT pseudo-code snippets for parametric and surface modelling are presented below.

Listing A.1: OCCT code for parametric blade construction.

```
#include OCCT_headers
TopoDS_Shape CreateParametricBlade(vector<Standard_Real>
   designParameters){
        //Populate 3-D sketch points
        vector<gp_Pnt> pointSketch1, pointSketch2, pointSketch3;
        for (int i = 0; i < designParameters.size/3; i+=3){
```

```
                Standard_Real X = designParameters[i];
                Standard_Real Y = designParameters[i+1];
                Standard_Real Z = designParameters[i+2];
                gp_Pnt controlPoint(X, Y, Z);
                pointSketch1.push_back(controlPoint);
        }

        Handle(Geom_BSplineCurve) curveSketch1 =
                new Geom_BSplineCurve(pointSketch1);

        //constructing topological edge from geometric curve
        TopoDS_Edge edgeSketch1 = BRepBuilderAPI_MakeEdge(
            curveSketch1);

        //same for two other sketches
        ...

        //OCCT lofting operation
        BRepOffsetAPI_ThruSections lofter;
        lofter.SetMaxSplineDegree(5);
        lofter.Add(edgeSketch1, edgeSketch2, edgeSketch3);
        lofter.Build();

        TopoDS_Shape blade;
        if (lofter.IsDone())
                blade = lofter.Shape();
        //Write resulting shape into step file
        StepWriter(blade, "blade.stp");

        return blade;
};
```

Listing A.1 uses OCCT to construct a parametric model of a blade. The model is built with the cross-sectional design approach (Subsec. A.1.1): three sketches/curves with pre-defined control points are used as an input in the lofting operation. The shape of the sketches and hence the blade is governed by the control point positions of the section curves. The resulting shape is stored to the STEP file.

The sequence of the C++ commands, in our case `CreateParametricBlade()` in Listing A.1, defines the parametrisation of the blade with certain input design parameters. However, the resulting STEP file stores only the final BRep: Topology object with corresponding Geometry/NURBS surfaces.

Listing A.2: STEP Reader.

```
#include "OCCT_headers.h"

Handle(Geom_BSplineSurface) GetBladeFirstSurface(){
        //Read Step File
        TopoDS_Shape blade = StepReader("blade.step");

        //Populate list of faces and edges
        TopTools_IndexedMapOfShape facemap, edgemap;
        TopExp::MapShapes(blade, TopAbs_FACE, facemap);
        TopExp::MapShapes(blade, TopAbs_EDGE, edgeMap);

        //Get topological face and corresponding geometric surface
        TopoDS_Face face1 = TopoDS::Face(facemap(1));
        Handle(Geom_BSplineSurface) bsplineSurf1
                                    = BRep_Tool::Surface(face1);
        return bsplineSurf1;
};
```

Listing A.2 presents OCCT code that reads a CAD model from a STEP file and explores its topological and geometrical entities. In the next Listing A.3, OCCT is used to access points on the surface of the model and modify one control point position, which in turn changes the geometry of the corresponding model.

Listing A.3: Surface Modification with OCCT.

```
#include "OCCT_headers.h"

void ExploreBladeSurface(){
 Handle(Geom_BSplineSurface)bsplineSurface=GetBladeFirstSurface();

 //parametric bounds of the surface
 Standard_Real u1, u2, v1, v2;
 bsplineSurface->Bounds(u1, u2, v1, v2);
 Standard_Real ustep = (u2-u1)/10.;
 //traversing all Cartesian points of the surface
 for (Standard_Real u = u1; u<u2; u+=ustep){
  for (Standard_Real v = v1; v<v2; v+=vstep){
        gp_Pnt Xs;
        bsplineSurface->D0(u,v,Xs);
        //do something with Xs
        }
 }
 //Updating Control Point position / surface modification
 int countU_ControlP = bsplineSurface->NbUPoles();
 int countV_ControlP = bsplineSurface->NbVPoles();

 gp_Pnt cp = bsplineSurface->Pole(countU_ControlP/2,
    countV_ControlP/2);
```

```
 Standard_Real displacement = 10;
 cp.SetZCoord(cp.CoordZ()+displacement);
 bsplineSurface->SetPole(countU_ControlP/2,countV_ControlP/2,cp);

};
```

In summary, OCCT presents a rich infrastructure for modelling of new components or manipulation of already existing shapes using C++. Similarly to the listings above, OCCT can be used to perform typical CAD operations for solid and surface modelling, as described in the previous sections. The Shape Optimisation Framework, developed in this thesis, uses OCCT algorithms for parametric modelling, standard CAD I/O, point inversion (point projections on the surface), edge/curve projection on face/surface, point calculation on the surface, surface-surface intersection and others. Additionally, the framework exploits derivatives of these algorithms available through Automatic Differentiation.

Appendix B

Getting Derivatives from OCCT Shapes: Code Snippets

Listing B.1: Overloading of multiplication operator in ADOL-C (vector mode).

```
class adouble{
double value;
double *ADvalue = new double[NUMBER_OF_INPUTS];
// multiplication overloading
inline myadouble operator * (const myadouble& a) const {
  myadouble tmp;
  tmp.value = value * a.value;
  for(size_t i = 0; i < NUMBER_OF_INPUTS; ++i)
    tmp.ADvalue[i] = ADvalue[i] * a.value + value * a.ADvalue[i];
  return tmp;
}
...
double* getADValue(){return ADValue;}
};
```

Listing B.2: Getting CAD derivatives on the surface point with diff. OCCT.

```
#include OCCT_headers (typedef adouble Standard_Real)
....
//Function constructs parametric blade as in the original OCCT
//See Listing A.1
TopoDS_Shape CreateParametricBlade(vector<Standard_Real>
   designParameters){...};

int n = designParameters.size();
vector<double> gradient(n);
for (int i = 0; i<designParameters.size(); i++){
```

```
        //seed design parameters
        designParameters[i].setADvalue(1);
        //Seeding with Cartesian unit vector:
        //designParameters.ADValues = [0,0,..,1,..,0]
        shape = CreateParametricBlade(designParameters);
        gp_Pnt Xs = shape->D0(u,v);
        gradient[i] = Xs.getADvalue();
        //unseed previous parameter
        designParameters[i].setADvalue(0);
}
```

Listing B.3: Recording trace/computational graph with ADOL-C.

```
int trace_tag = 1;
void GenerateTrace(){
        int n = designParameters.size() ; // number of inputs
        int m = surfacePoints().size() * 3;// number of outputs
        double *output = new double[m];

        trace_on(trace_tag);    // begin tracing
        //select input parameters
        for(int i = 0; i < n; i++)
                designParameters[i] <<= designParameters[i].
                   getValue();

        //See Listing A.1
        TopoDS_Shape blade =
        CreateParametricBlade(designParameters);

        int j = 0;
        for (int i = 0; i < m; i++){
                gp_Pnt p = shape->D0(surfacePoint);
                //select output variables
                p.X() >>= output[j]; j++;
                p.Y() >>= output[j]; j++;
                p.Z() >>= output[j]; j++;
                }
        trace_off(); // stop tracing
}
```

In Listing B.3, the special symbols <<= and >>= are used by ADOL-C to indicate the input and output variables in the trace.

Listing B.4: Getting CAD derivatives from traces (tapes) with forward and reverse AD modes.

```
vector<double> cad_grad_rev(n), cost_sens_rev(n);
vector<double> cad_grad_forw(m);
GenerateTrace();

double* independent = new double[n];
for (auto i = 0; i < n; i++)
        independent[i] = designParameters[i].getValue();

//tangent and adjoint seed vectors
double* tangentVector = [0,0...,1,...0];  //size(n)
double* adjointVector  = [0,0,1,0];   // size(m)

//forward or reverse CAD gradients computation
jac_vec(trace_tag,m,n,independent,tangentVector,cad_grad_forw);
vec_jac(trace_tag,m,n,independent,adjointVector,cad_grad_rev);
...
//reverse mode aerodynamic cost sensitivity
adjointVector  = cfdSens;   // size(m)
vec_jac(trace_tag,m,n,independent,adjointVector,cost_sens_rev);

//use gradients for optimisation
...
```

Listing B.5: Getting CAD derivatives on the surface point with the block-vector AD mode.

```
#include OCCT_headers (typedef adouble Standard_Real)
#NUMBER_OF_DIRECTIONS = p

vector<Standard_Real> designParameters = getDesignParameters();
//size (n);
vector<double> gradient(n);
int numberOfBlocks  = ceil(n / p);
int currentBlock = 0;
while (currentBlock < numberOfBlocks){
    //seed block
    for (int i = 0; i<p; i++){
       designParameters[currentBlock * p + i].setADvalue(i,1);
    }
    //Creation of parametric blade and cylinders
    //See Listing A.1
    TopoDS_Shape blade = CreateParametricBlade(designParameters);
    TopoDS_Shape cylinders = CreateCylinders(designParmeters);

    //Boolean Union of the blade and cylinders
    BRepAlgoAPI_Fuse unionBuilder(blade,cylinders);
    unionBuilder.Build();
```

```
    TopoDS_Shape shape = unionBuilder.Shape();

    gp_Pnt Xs = shape->D0(u,v);
    gradient[currentBlock * p + i] = Xs.getADvalue(i);
    //unseed previous block
}
//use gradient
....
```

Appendix C

CAD Collisions Detection

The differentiated OCCT can be used to detect collisions between different CAD shapes and include such constraints into the optimisation loop. Figure C.1 il-

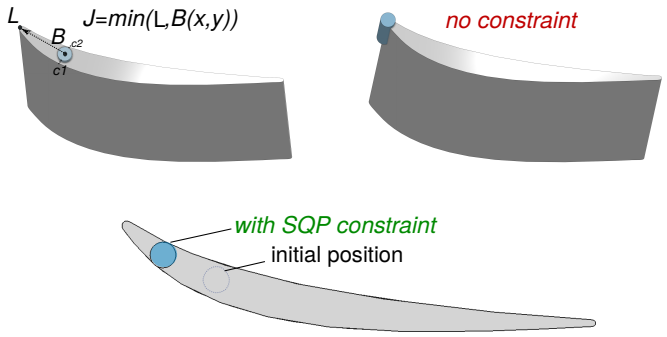

Figure C.1: Cylinder-to-LE distance minimisation: without (top) and with (bottom) collision constraints.

lustrates a corresponding optimisation test-problem: the distance J between the centre of the cylindrical shape B and the TUB blade's (Sec. 4.1) leading edge L has to be minimised. In this case, the coordinates (x, y) of the cylinder centre are considered as design parameters. Additionally, the cylinder needs to remain inside the blade (no collision with the exterior of the blade is permitted). Naturally, the unconstrained gradient-based optimisation places the cylinder directly at the leading edge ($J = 0$) but violates the constraint. To avoid this, constraint

functions $c1, c2$ are implemented to measure the distance between the cylinder and pressure/suction sides of the blade. The use of the constraints ($c1 > 0, c2 > 0$) and their gradients in the SLSQP optimisation algorithm leaves the cylinder inside the blade and moves it closer to the leading edge.

In a similar fashion, the flexibility of OCCT allows users to define cases-specific constraints (distance metrics, intersection areas between two shapes) and account for the collisions between CAD parts. The approach can be also applied for the aerodynamic optimisation applications.

Appendix D

Author's Publications

Journal papers

1. Mykhaskiv O., Banovic M., Auriemma S., Mohanamuraly P., Walther A., Legrand H., Müller J.D., "*NURBS-based and parametric-based shape optimisation with differentiated CAD Kernel*", Computer-Aided Design and Applications, 2018.

2. Banovic M., Mykhaskiv O., Auriemma S., Walther A., Legrand H., Müller J.D., "*Algorithmic differentiation of the OpenCascade Technology CAD kernel and its coupling with adjoint CFD solver*", Optimization Methods and Software, 2018.

3. Xu S., Timme S., Mykhaskiv O., Müller J.D., "*Wing-body junction optimisation with CAD-based parametrisation including a moving intersection*", Aerospace Science and Technology, 2017.

Conference papers

1. Auriemma S., Banovic M., Mykhaskiv O., Walther A., Müller J.D., "*Applications of differentiated CAD kernel in gradient-based aerodynamic shape optimisation*", AIAA Energy and Propulsion Forum, 2018.

2. Müller J.D., Hueckelheim J., Mykhaskiv O., "*STAMPS: a finite-volume solver framework for adjoint codes derived with source transformation AD*", AIAA Multidisciplinary Analysis and Optimization Conference, 2018.

3. Mykhaskiv O., Mohanamuraly P., Müller J.D., Xu S., Timme S., "*CAD-based shape optimisation of the NASA CRM wing-body intersection using differentiated CAD kernel*", 35th AIAA Applied Aerodynamics Conference, 2017.

4. Auriemma S., Mykhaskiv O., Banovic M., Walther A., Legrand H., Müller J.D., "*Optimisation of a U-bend using CAD-based adjoint method with differentiated CAD kernel*", ECCOMACS Congress, 2016.

Bibliography

[1] Airbus presents the A380plus. http://www.airbus.com/newsroom/press-releases/en/2017/06/airbus-presents-the-a380plus.html. [Online; accessed 1-April-2018].

[2] CAESES: Robust variable geometry for CFD. https://www.caeses.com/. [Online; accessed 1-April-2018].

[3] FeeeCAD: Parametric 3D Modeller. https://www.freecadweb.org/. [Online; accessed 4-April-2018].

[4] GridPro: platform with advanced meshing algorithm underhood. https://www.gridpro.com. [Online; accessed 12-February-2018].

[5] OnShape: Cloud-based 3D CAD Modeller. https://www.onshape.com/. [Online; accessed 1-May-2018].

[6] OpenCASCADE: Core Technology. https://www.opencascade.com/content/core-technology. [Online; accessed 24-April-2019].

[7] OpenCASCADE Development Portal. https://dev.opencascade.org/. [Online; accessed 5-September-2018].

[8] OpenCASCADE: Surface Filling. https://www.opencascade.com/doc/occt-6.9.1/refman/html/class_b_rep_offset_a_p_i___make_filling.html. [Online; accessed 4-April-2018].

[9] STAR-CCM+. https://mdx.plm.automation.siemens.com/star-ccm-plus. [Online; accessed 2-April-2018].

[10] TU Munich DrivAer Model. http://www.aer.mw.tum.de/en/research-groups/automotive/drivaer/. [Online; accessed 25-July-2018].

[11] AIAA: 6th CFD Drag Prediction Workshop. https://aiaa-dpw.larc.nasa.gov/, 2016. [Online; accessed 10-January-2019].

[12] AboutFlow Optimisation benchmark: TU Berlin TurboLab Stator. http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase3/, 2017. [Online; accessed 10-May-2018].

[13] AboutFlow Optimisation benchmark: TU Munich DrivAer Model. http://aboutflow.sems.qmul.ac.uk/events/munich2016/benchmark/testcase4/, 2017. [Online; accessed 10-May-2018].

[14] Applications of Automatic Differentiation. http://www.autodiff.org/?module=Applications, 2018. [Online; accessed 3-April-2018].

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. TensorFlow: large-scale machine learning on heterogeneous systems.

[16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[17] B. M. Adams, W. Bohnhoff, K. Dalbey, J. Eddy, M. Eldred, D. Gay, K. Haskell, P. D. Hough, and L. P. Swiler. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.0 user's manual. *Sandia National Laboratories, Tech. Rep. SAND2010-2183*, 2009.

[18] D. Agarwal, C. Kapellos, T. T. Robinson, and C. G. Armstrong. Using parametric effectiveness for efficient CAD-based automotive design driven by adjoint sensitivity maps. In *EUROGEN 2017*.

[19] M. Aissa, T. Verstraete, and C. Vuik. Toward a GPU-aware comparison of explicit and implicit CFD simulations on structured meshes. *Computers & Mathematics with Applications*, 74(1):201–217, 2017.

[20] T. A. Albring, M. Sagebaum, and N. R. Gauger. Efficient aerodynamic design using the discrete adjoint method in SU2. In *17th AIAA/ISSMO multidisciplinary analysis and optimization conference*, page 3518, 2016.

[21] P. P. Alexias and E. de Villiers. Gradient projection, constraints and surface regularization methods in adjoint shape optimization. In *Evolutionary and Deterministic Methods for Design Optimization and Control With Applications to Industrial and Societal Problems*, pages 3–17. Springer, 2019.

[22] S. R. Allmaras and F. T. Johnson. Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model. In *Seventh international conference on computational fluid dynamics (ICCFD7)*, pages 1–11, 2012.

[23] S. Auriemma, M. Banovic, O. Mykhaskiv, H. Legrand, J.-D. Müller, and A. Walther. Optimisation of a U-bend using CAD-based adjoint method with differentiated CAD kernel. In *ECCOMAS Congress*, 2016.

[24] M. Banovic, O. Mykhaskiv, S. Auriemma, A. Walther, H. Legrand, and J. D. Müller". Automatic differentiation of the OpenCascade Technology CAD system and its coupling with an adjoint CFD solver. In *Optimization Methods and Software*, 2017.

[25] J. T. Batina. Unsteady Euler airfoil solutions using unstructured dynamic meshes. 28(8):1381–1388, 1990.

[26] A. G. Baydin and B. A. Pearlmutter. Diffsharp: Automatic differentiation library. In *International Conference on Machine Learning (ICML) Workshop on Machine Learning Open Source Software 2015: Open Ecosystems, Lille, France, July 10, 2015*, 2015.

[27] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research (JMLR)*, 18(153):1–43, 2018.

[28] R. Bénière, G. Subsol, G. Gesquière, F. Le Breton, and W. Puech. A comprehensive process of reverse engineering from 3D meshes to CAD models. *Computer-Aided Design*, 45(11):1382–1393, 2013.

[29] C. Bischof, A. Carle, P. Hovland, P. Khademi, and A. Mauer. *ADiFor 2.0 Users' Guide*. Argonne National Laboratory, 1998.

[30] C. H. Bischof, H. Bucker, B. Lang, A. Rasch, and A. Vehreschild. Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In *Source Code Analysis and Manipulation, 2002. Proceedings. Second IEEE International Workshop on*, pages 65–72. IEEE, 2002.

[31] D. Brujic, M. Ristic, M. Mattone, P. Maggiore, and G. P. De Poli. CAD-based shape optimization for gas turbine component design. *Structural and Multidisciplinary Optimization*, 41(4):647–659, 2010.

[32] F. Christakopoulos. *Sensitivity computation and shape optimisation in aerodynamics using the adjoint methodology and Automatic Differentiation*. PhD thesis, Queen Mary University of London, 2013.

[33] B. Christianson. Reverse accumulation and implict functions. *Optimization Methods and Software*, 9(4):307–322, 1998.

[34] F. Courty, A. Dervieux, B. Koobus, and L. Hascoët. Reverse Automatic Differentiation for optimum design: from adjoint state assembly to gradient computation. *Optimization Methods and Software*, 18(5):615–627, 2003.

[35] N. Cumpsty and J. Horlock. Averaging non-uniform flow for a purpose. In *ASME Turbo Expo 2005: Power for Land, Sea, and Air*, pages 1–14. American Society of Mechanical Engineers Digital Collection, 2005.

[36] J. Dannenhoffer and R. Haimes. Using design-parameter sensitivities in adjoint-based design environments. In *55th AIAA Aerospace Sciences Meeting*, page 0139, 2017.

[37] A. De Boer, M. S. Van der Schoot, and H. Bijl. New method for mesh moving based on radial basis function interpolation. In *In ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics*, 2006.

[38] B. Diskin and J. L. Thomas. Comparison of node-centered and cell-centered unstructured finite-volume discretizations: inviscid fluxes. *AIAA journal*, 49(4):836–854, 2011.

[39] R. Duvigneau. Adaptive parameterization using free-form deformation for aerodynamic shape optimization. Report 5949, INRIA, 2006.

[40] R. P. Dwight. Robust mesh deformation using the linear elasticity equations. In *Computational Fluid Dynamics 2006*, pages 401–406. Springer, 2009.

[41] T. D. Economon. *Optimal Shape Design Using an Unsteady Continuous Adjoint Approach*. PhD thesis, Stanford University, 2014.

[42] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso. SU2: an open-source suite for multiphysics simulation and design. *AIAA Journal*, 54(3):828–846, 2015.

[43] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3D object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[44] P. Fayolle and A. Pasko. An evolutionary approach to the extraction of object construction trees from 3D point clouds. *Computer-Aided Design*, 74:1–17, 2016.

[45] A. Fluent. 14.0 Theory Guide. *ANSYS inc*, 0:390–1, 2011.

[46] S. A. Forth and T. P. Evans. Aerofoil optimisation via AD of a multigrid cell-vertex Euler flow solver. In *Automatic differentiation of algorithms*, pages 153–160. Springer, 2002.

[47] R. Giering and T. Kaminski. Applying TAF to generate efficient derivative code of Fortran 77-95 programs. *PAMM*, 2(1):54–57, 2003.

[48] M. Giles, D. Ghate, and M. Duta. Using Automatic Differentiation for Adjoint CFD code development. In B. Uthup, S.-P. Koruthu, R.-K. Sharma, and P. Priyadarshi, editors, *Recent Trends in Aerospace Design and Optimization*. Tata-McGraw Hill, New Delhi, 2005. Post-SAROD-2005, Bangalore, India.

[49] M. B. Giles, M. C. Duta, J.-D. Müller, and N. A. Pierce. Algorithm developments for discrete adjoint methods. *AIAA Journal*, 41(2):198–205, 2003.

[50] J. Gräsel, A. Keskin, M. Swoboda, H. Przewozny, and A. Saxer. A full parametric model for turbomachinery blade design and optimisation. In *ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 907–914. American Society of Mechanical Engineers, 2004.

[51] A. Griewank, D. Juedes, and J. Utke. ADOL-C: A package for the Automatic Differentiation of Algorithms written in C/C++. 1996.

[52] A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, PA, Second edition, 2008.

[53] M. Gugała. *Output-based mesh adaptation using geometric multi-grid for error estimation*. PhD thesis, School of Engineering and Materials Science, Queen Mary University of London, London, UK, 2018. to be submitted.

[54] L. Hascoët and V. Pascual. The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. *ACM Transactions On Mathematical Software*, 39(3), 2013.

[55] A. Haselbacher, J. J. McGuirk, and G. J. Page. Finite volume discretization aspects for viscous flows on mixed unstructured grids. *AIAA journal*, 37(2):177–184, 1999.

[56] A. I. Heft, T. Indinger, and N. A. Adams. Experimental and numerical investigation of the DrivAer model. In *ASME 2012 Fluids Engineering Division Summer Meeting collocated with the ASME 2012 Heat Transfer Summer Conference and the ASME 2012 10th International Conference on Nanochannels, Microchannels, and Minichannels*, pages 41–51. American Society of Mechanical Engineers Digital Collection, 2012.

[57] J. C. Hückelheim. *Discrete adjoints on many cores: Algorithmic Differentiation of accelerated fluid simulations*. PhD thesis, Queen Mary University of London, 2017.

[58] J. F. D. III and R. Haimes. Design sensitivity calculations directly on CAD-based geometry. In *53rd AIAA Aerospace Sciences Meeting*, page 1370, 2015.

[59] D. Jacobsen, J. Thibault, and I. Senocak. An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, page 522, 2010.

[60] S. Jakobsson and O. Amoignon. Mesh deformation using radial basis functions for gradient-based aerodynamic shape optimization. *Computers and Fluids*, 36(6):1119 – 1136, 2007.

[61] A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3:233–260, 1988.

[62] A. Jameson. Computational aerodynamics for aircraft design. *Science*, 245(4916):361–371, 1989.

[63] A. Jameson. Optimum aerodynamic design using CFD and control theory. *AIAA paper*, 1729:124–131, 1995.

[64] A. Jameson. Optimum aerodynamic design using control theory. In M. Hafez and K. Oshima, editors, *Computational Fluid Dynamics Review 1995*, pages 492–528. John Wiley & Sons, 1995.

[65] A. Jameson and A. Vassberg. Studies of alternative numerical optimization methods applied to the brachistochrone problem. *Computational Fluid Dynamics Journal*, 9(3), 2000.

[66] H. Jasak, A. Jemcov, Z. Tukovic, et al. OpenFOAM: A C++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik, Croatia, 2007.

[67] A. Jaworski, J.-D. Müller, and J. Rokicki. One-shot optimisation with grid adaptation using adjoint sensitivities. In *Proceedings ECCOMAS 2012*, pages 3685–3693, 2012.

[68] R. Jesudasan, X. Zhang, M. Gugala, and J.-D. Müller. CAD-free vs. CAD-based parametrisation method in adjoint-based aerodynamic shape optimisation. In *Proceedings ECCOMAS 2016*, 2016.

[69] R. Jesudasan, X. Zhang, and J.-D. Müller. Adjoint optimisation of internal turbine cooling channel using NURBS-based automatic and adaptive parametrisation method. In *Proceedings of the fifth ASME Gas Turbine India Conference, December 7-8*, 2017.

[70] E. Jones, T. Oliphant, and P. Peterson. SciPy: open source scientific tools for Python. Technical report, https://www.scipy.org/, 2014.

[71] D. Kapsoulis, K. Tsiakas, X. Trompoukis, V. Asouti, and K. Giannakoglou. Evolutionary multi-objective optimization assisted by metamodels, kernel PCA and multi-criteria decision making techniques with applications in aerodynamics. *Applied Soft Computing*, 64:1–13, 2018.

[72] G. K. Karpouzas, E. M. Papoutsis-Kiachagias, T. Schumacher, E. de Villiers, K. C. Giannakoglou, and C. Othmer. Adjoint optimization for vehicle external aerodynamics. 2016.

[73] D. Kraft. A software package for sequential quadratic programming. *Forschungsbericht Deutsche Forschungs und Versuchsanstalt fur Luft und Raumfahrt*, 1988.

[74] L. Langston. Secondary flows in axial turbines—a review. *Annals of the New York Academy of Sciences*, 934(1):11–26, 2001.

[75] L. Lapworth. Hydra-CFD: a framework for collaborative CFD development. In *International Conference on Scientific and Engineering Computation (IC-SEC), Singapore, June*, volume 30, 2004.

[76] C. L. Lawson and R. J. Hanson. *Solving least squares problems*, volume 15. Siam, 1995.

[77] M.-S. Liou and C. J. Steffen. A new flux splitting scheme. *Journal of Computational Physics*, 107(1):23–39, 1993.

[78] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[79] J. E. Lugo, S. M. Batill, and L. Carlson. Modeling product form preference using Gestalt principles, semantic space, and Kansei. *ASME Paper No. DETC2012-70434*, pages 529–539, 2012.

[80] Z. Lyu, Z. Xu, and J. Martins. Benchmarking optimization algorithms for wing aerodynamic design optimization. In *Proceedings of the 8th International Conference on Computational Fluid Dynamics, Chengdu, Sichuan, China*, volume 11, 2014.

[81] D. Maclaurin, D. Duvenaud, and R. P. Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, 2015.

[82] M. J. Martín Burgos. *NURBS-based geometry parameterization for aerodynamic shape optimization.* PhD thesis, Polytechnic University of Madrid, 2015.

[83] M. J. Martín Burgos, M. Cordera-Gracia, and M. Gomez. Adaptation of the computational grid to a moving wing-fuselage intersection via NURBS and radial basis. 2014.

[84] J. R. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software (TOMS)*, 29(3):245–262, 2003.

[85] J. Mihalyovics, C. Brueck, D. Peitsch, I. Vasilopoulos, and M. Meyer. CAD-based aerodynamic optimization of a compressor stator using conventional and adjoint-driven approaches. In *Proceedings of Turbomachinery Technical Conference and Exposition TURBO EXPO 2018*, 2018.

[86] P. Mohanamuraly. *Fast Adjoint-assisted Multilevel Multifidelity Method for Uncertainty Quantification of the Aleatoric Kind*. PhD thesis, Queen Mary University of London, 2019.

[87] J.-D. Müller and P. Cusdin. On the performance of discrete adjoint CFD codes using automatic differentiation. *International Journal for Numerical Methods in Fluids*, 47(6-7):939–945, 2005.

[88] J.-D. Müller, P. Mohanamuraly, S. Xu, J. Hückelheim, and M. Gugala. STAMPS: a finite-volume solver framework for adjoint codes derived with source-transformation AD. *AIAA-CP XX-2018*, 2018. submitted.

[89] O. Mykhaskiv, M. Banovic, S. Auriemma, A. Walther, and J.-D. Mueller. NURBS-based and parametric-based shape optimisation with differentiated CAD kernel. *Computer-Aided Design and Applications*, pages 1–11, 2018.

[90] O. Mykhaskiv, P. Mohanamuraly, J.-D. Müller, S. Xu, and S. Timme. CAD-based shape optimisation of the NASA CRM wing-body intersection using differentiated CAD kernel. In *35th AIAA Applied Aerodynamics Conference*, 2017.

[91] S. Nadarajah and A. Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. *AIAA CP-00-0667*, 2000.

[92] D. Nagy, D. Zhao, and D. Benjamin. Nature-based hybrid computational geometry system for optimizing component structure. In *Humanizing Digital Reality*, pages 167–176. Springer, 2018.

[93] U. Naumann. *The art of differentiating computer programs: an introduction to algorithmic differentiation*, volume 24. Siam, 2012.

[94] U. Naumann, J. Lotz, K. Leppkes, and M. Towara. Algorithmic differentiation of numerical methods: Tangent and adjoint solvers for parameterized

systems of nonlinear equations. *ACM Trans. Math. Softw.*, 41(4):26:1–26:21, Oct. 2015.

[95] C. Othmer and T. Grahs. CFD topology and shape optimization with adjoint methods. Technical Report VDI-Berichte Nr 1967, VDI, 2006.

[96] C. Othmer, T. W. Lukaczyk, P. Constantine, and J. J. Alonso. On active subspaces in car aerodynamics. In *17th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, page 4294, 2016.

[97] F. Palacios, M. R. Colonno, A. C. Aranake, A. Campos, S. R. Copeland, T. D. Economon, A. K. Lonkar, T. W. Lukaczyk, T. W. Taylor, and J. J. Alonso. Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design. *AIAA Paper*, 287:2013, 2013.

[98] E. Papoutsis-Kiachagias, N. Magoulas, J. Mueller, C. Othmer, and K. Giannakoglou. Noise reduction in car aerodynamics using a surrogate objective function and the continuous adjoint method with wall functions. *Computers & Fluids*, 122:223–232, 2015.

[99] E. M. Papoutsis-Kiachagias and K. C. Giannakoglou. Continuous adjoint methods for turbulent flows, applied to shape and topology optimization: Industrial applications. *Archives of Computational Methods in Engineering*, 23(2):255–299, 2016.

[100] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic Differentiation in PyTorch. 2017.

[101] R. E. Perez, P. W. Jansen, and J. R. Martins. pyOpt: a Python-based object-oriented framework for nonlinear constrained optimization. *Structural and Multidisciplinary Optimization*, 45(1):101–118, 2012.

[102] L. Piegl and W. Tiller. *The NURBS book*. Springer Science & Business Media, 2012.

[103] O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64:97–110, 1974.

[104] O. Reutter, M. Rozanski, A. Hergt, and E. Nicke. Advanced endwall contouring for loss reduction and outflow homogenization for an optimized compressor cascade. *International journal of turbomachinery, Propulsion and Power*, 2(1):1, 2017.

[105] T. Robinson, C. Armstrong, H. Chua, C. Othmer, and T. Grahs. Optimizing parameterized CAD geometries using sensitivities based on adjoint functions. *Computer-Aided Design & Applications*, 9(3):253–268, 2012.

[106] P. Roe. Approximate Riemann solvers, parameter vectors and difference schemes. *Journal of Computational Physics*, 43(2), 1981.

[107] M. Sagebaum, T. Albring, and N. R. Gauger. High-performance derivative computations using CoDiPack. *arXiv preprint arXiv:1709.07229*, 2017.

[108] J. Samareh. Aerodynamic shape optimization based on free-form deformation. *AIAA*, paper 4630:1–13, 2004. DOI:10.2514/6.2004-4630.

[109] S. Schlenkrich, A. Walther, N. Gauger, and R. Heinrich. Differentiating fixed point iterations with ADOL-C: Gradient calculation for fluid dynamics. In *Proceedings of High Performance Computing*, Hanoi, March 6-10 2006. submitted.

[110] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[111] S. Shahpar and L. Lapworth. PADRAM: Parametric design and rapid meshing system for turbomachinery optimisation. In *ASME Turbo Expo 2003,*

Collocated with the 2003 International Joint Power Generation Conference, pages 579–590. American Society of Mechanical Engineers, 2003.

[112] A. Stück and T. Rung. Adjoint complement to viscous finite-volume pressure-correction methods. *Journal of Computational Physics*, 248:402 – 419, 2013.

[113] I. S. Torreguitart, T. Verstraete, and L. Mueller. CAD and adjoint-based multipoint optimization of an axial turbine profile. In *Evolutionary and Deterministic Methods for Design Optimization and Control With Applications to Industrial and Societal Problems*, pages 35–46. Springer, 2019.

[114] M. Towara and U. Naumann. A discrete adjoint model for OpenFOAM. *Procedia Computer Science*, 18:429–438, 2013.

[115] M. Towara, M. Schanen, and U. Naumann. MPI-parallel discrete adjoint OpenFOAM. *Procedia Computer Science*, 51:19–28, 2015.

[116] J. Utke, U. Naumann, M. Fagan, N. Tallent, M. Strout, P. Heimbach, C. Hill, and C. Wunsch. OpenAD: A modular open-source tool for automatic differentiation of Fortran codes. *ACM Transactions on Mathematical Software (TOMS)*, 34(4):18, 2008.

[117] L. Uyttersprot. Inverse Distance Weighting Mesh Deformation. Master's thesis, Delft University of Technology, 2014.

[118] A. Valencia-Romero and J. E. Lugo. Quantification of symmetry, parallelism, and continuity as continuous design variables for three-dimensional product representations. In *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2016.

[119] I. Vasilopoulos, P. Flassig, and M. Meyer. CAD-based aerodynamic optimization of a compressor stator using conventional and adjoint-driven approaches.

In *ASME Turbo Expo 2017: Turbomachinery Technical Conference and Exposition*. American Society of Mechanical Engineers, 2017.

[120] J. Vassberg, M. Dehaan, M. Rivers, and R. Wahls. Development of a common research model for applied CFD validation studies. In *26th AIAA Applied Aerodynamics Conference*, page 6919, 2008.

[121] T. Verstraete. CADO: a Computer-Aided Design and Optimization tool for turbomachinery applications. In *2nd Int. Conf. on Engineering Optimization, Lisbon, Portugal, September*, pages 6–9, 2010.

[122] T. Verstraete, F. Coletti, J. Bulle, T. Vanderwielen, and T. Arts. Optimization of a U-bend for minimal pressure loss in internal cooling channels — Part I: Numerical Method. *Journal of Turbomachinery*, 135(5):051015, 2013.

[123] A. Walther, A. Griewank, and O. Vogel. ADOL-C: Automatic differentiation using operator overloading in C++. *Proc. App. Math. Mech*, 2(1):41–44, 2004.

[124] S. Weickgenannt and S. Harries. Parametric-adjoint Optimization using STAR-CCM+ and CAESES, 2016.

[125] J. A. Witteveen. Explicit and robust inverse distance weighting mesh deformation for CFD. In *48th AIAA Aerospace Sciences Meeting*, volume 165, 2010.

[126] S. Xu, W. Jahn, and J.-D. Müller. CAD-based shape optimisation with CFD using a discrete adjoint. *International Journal for Numerical Methods in Fluids*, 74(3):153–68, 2013.

[127] S. Xu, D. Radford, M. Meyer, and J.-D. Müller. CAD-based adjoint shape optimisation of a one-stage turbine with geometric constraints. ASME Turbo Expo, June 2015. GT2015-42237.

[128] S. Xu, D. Radford, M. Meyer, and J.-D. Müller. Stabilisation of discrete steady adjoint solvers. *Journal of Computational Physics*, 299:175–195, 2015.

[129] S. Xu, S. Timme, O. Mykhaskiv, and J.-D. Müller. Wing-body junction optimisation with CAD-based parametrisation including a moving intersection. *Aerospace Science and Technology*, 2017.

[130] X. Zhang. *CAD-based geometry parametrisation for shape optimisation using Non-uniform Rational B-splines*. PhD thesis, Queen Mary University of London, 2018.